Por Castro Pippo, Juan Manuel 12/08/17

Índice

Sockets

Teoría (17) (4/9)

(5) ¿Qué diferencias existen entre una comunicación UDP y TCP? Describa la forma de enviar datos y qué funciones requieren ser invocadas en cada caso. Describa un uso adecuado para cada uno de ellos.

Característica	TCP	UDP
Orientado a la conexión	Sí	No
Utilizado en tiempo real	No	Sí
Recuperación de errores	Sí	No
Control de flujo	Sí	No

UDP se utiliza mayormente para transmitir audio y video, y para conexiones de juegos en red, ya que lo primordial en esos casos es la velocidad y no es tan importante si se pierde algún paquete. Asimismo se utiliza para enviar mensajes de broadcast en una red.
TCP se utiliza cuando la información a transmitir debe llegar por completo y se decide resignar un poco de velocidad a cambio de la confiabilidad. Es un protocolo fiable que se encarga de asegurar el correcto orden de llegada de los paquetes y la retransmisión ante una eventual pérdida. Por este motivo las transmisiones FTP, por ejemplo, utilizan TCP.
Recordemos que es orientado a la conexión; lo que implica que determina de alguna manera si el otro extremo está presente.

¿Qué propósito tiene la función socket?

Explique el propósito y uso de la función BIND en una comunicación UDP y en una TCP.

(3) ¿Cual es el uso y proposito de la función LISTEN en una comunicación TCP y en una UDP?¿qué parámetros tiene?¿para que sirven?
<simil>
(2) ¿Qué función conoce para limitar la cola de conexiones entrantes a un socket? ¿Cómo se utiliza?

(3) Explique la función Accept haciendo referencia mínimamente a su formato, propósito y tipo de llamada/retorno.¿Qué parámetros posee? ¿Para qué sirven?

Explique el uso de la función RECEIVE en comunicaciones orientadas a la conexión. Describa sus parámetros y su comportamiento en relación a la cantidad de bytes que “lee”.

Escriba el .H de una bilbioteca de funciones ANSI C para manejar una comunicación TCP.

¿Qué significa que una función (de socket) es blocante?¿Cómo subsanaría esa limitación?
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 muertos de los bloqueos.

Comunicaciones: ¿Qué formas conoce para determinar cuando se recibió un paquete de datos completo?
Una manera es que los paquetes tengan una longitud definida por protocolo.
Otra manera esque al inicio del paquete se reserven bytes para indicar el tamaño del paquete. (En caso de paquetes de texto queda medio feo porque se mezcla binario para la longitud con texto que es el mensaje).
Otra manera es utilizar un byte reservado que se utilice para indicar la finalizacion de un paquete.

Suponga que dos aplicaciones (A y B) se encuentran comunicadas por un socket TCP. La aplicación A envía a la B un paquete de 123 bytes. ¿Como puede recibir la aplicación B esos datos? En ún solo paquete?...Partidos en 2?.. en 3?....
El la aplicación A envíe un paquete de 123bytes no implica que la aplicación B va a recibir un único paquete de 123bytes, el mismo puede llegar fraccionado en hasta 123 paquetes de 1byte. Lo que si asegura protocolo TCP esque van a llegar en orden.

Práctica / Programar (21) (8/16)

Ver Resueltos en ../Ejercicios Sockets/<nroEjercicio>/

Ejercicios de Cliente

1) Escriba una pequeña aplicación C/C++ que se conecte al puerto 2000 del equipo 192.168.1.2, transmita la cadena “hola” y se cierre ordenadamente sin esperar respuesta alguna.

2) Implemente un programa C que reciba por línea de comandos 3 parámetros: IP, PUERTO y ARCHIVO; y que transmita el archivo binario <ARCHIVO> al puerto <PUERTO> de la dirección <IP>, utilizando protocolo TCP/IP. No valide errores.

3) Escriba un programa que, recibiendo por línea de comandos una IP y un PUERTO, se conecte al puerto PUERTO del ordenador con ip IP y transmita la cadena “Aquí estoy”. El programa debe terminar cuando reciba “Yo También”.
<simil>
Escriba un programa que, recibiendo por línea de comandos una IP y un PUERTO, se conecte al puerto PUERTO del ordenador con ip IP y transmita la cadena “Me reporto”. El programa debe terminar cuando reciba “Ok”.

4) Implemente un programa que reciba por línea de comandos una dirección IP y un PUERTO; y transmita 50 paquetes de 10 bytes aleatorios a esa dirección/puerto a intervalos de 2 segundos.
<simil>
Escriba un programa que, recibiendo por línea de comandos una IP y un PUERTO, se conecte al puerto PUERTO del ordenador con ip IP y transmita 20 cadenas “hola” a espacios de 15 segundos aproximadamente.
<simil>
Escriba un programa que cada 3 segundos envíe un byte 'X' al puerto TCP 1433 del equipo identificado con la IP recibida por línea de comandos. El programa debe terminar ordenadamente luego de enviar 300 bytes.

5) Escriba una aplicación C que solicite una dirección IP, un puerto y un mensaje (string) por teclado. El programa debe conectarse al destino indicado, transmitir el mensaje y cerrar las conexiones ordenadamente antes de terminar.

Implemente un programa que reciba por línea de comandos una dirección IP y un puerto; y transmita 50 paquetes de 10 bytes aleatorios a esa dirección/puerto antes de terminar ordenadamente.

Escriba un programa que reciba por línea de comandos una IP y un PUERTO de conexión, se conecte al destino indicado y transmita los paquetes ingresados por teclado (cada línea ingresada por teclado se considerará un paquete). El programa debe terminar ordenadamente cuando se ingrese “FIN”, debiéndose este paquete ser transmitido.

Escriba el programa que, recibiendo por línea de comandos un PUERTO de escucha y una IP y PUERTO de conexión, escuche conexiones TCP en el puerto PUERTO de escucha y retransmita todo lo recibido a IP-PUERTO de conexión; y que devuelva a PUERTO todo lo recibido de la citada IP-PUERTO.

Ejercicios de Servidor

1) Escriba un pequeño programa que, sin validar errores, acepte una única conexión TCP en el puerto 1972 y reciba paquetes de no mas de 600 bytes, terminado en '0x00'. Cada paquete recibido debe ser devuelto con todos sus bits negados. Terminar al recibir un paquete nulo.

2) Escriba un programa que escuche conexiones TCP en el puerto 3280 y guarde en disco con nombres correlativos (1.html, 2.html , ...) cada uno de los HTML recibidos.
<simil>
Escriba un programa que escuche conexiones TCP en el puerto 1180 y guarde en disco con nombres correlativos (1.txt, 2.txt, ...) cada una de las páginas HTML recibidas (1 página por conexión).

3) Escriba un programa que acepte conexiones por el puerto 2283. En cada conexión abierta debe esperar paquetes de datos de 513 bytes y devolverlos al emisor. El programa debe terminar ordenadamente al recibir algún paquete de 513 letras 'Q'.

Escriba un programa que reciba por línea de comandos un Puerto y una IP. El programa debe imprimir en stdout todo lo recibido (aceptar una unica conexión a la vez). Si transcurren 3 segundos sin recibir nada debe finalizar.

Escriba un programa (desde la inicialización hasta la liberación de los recursos) que reciba paquetes de 11 bytes por el puerto TCP 802 y los imprima por pantalla. Al recibir el byte ‘0xCC’ debe cerrarse devolviendo al S.O. la cantidad de paquetes recibidos. No considere errores.

Implemente un programa que acepte conexiones por el puerto 5000 (de a 1 por vez). Al recibir una conexión debe transmitir el mensaje “error” y cerrar la conexión. El programa debe finalizar luego de haber atendido 10 conexiones.

Escriba un programa C que esuche por el puerto indicado en la línea de comandos y retorne todo el contenido recibido. El mismo debe aceptar múltiples conexiones y debe terminar ordenadamente cuando se reciba un 00 01 02.

Otros

Defina la función:
int RecibirYAlmacenar(Socket *S);
Esta función debe recibir paquetes de 256 bytes y mandarlos a almacenar valiéndose de la función Almacenar(const char *,int). La recepción debe terminarse cuando Almacensar devuelva algún valor negativo.

Hilos / Threads

Teoría (27) (9/10)

Los contenedores STL (como std::list, std::vector, etc) no son thread-safe. Ello implica que siempre hay que sincronizar operaciones sobre los contenedores con un mutex.

(2) ¿Qué es un programa multi-hilo (MultiThread)? ¿Cuales son sus ventajas frente a uno que no lo es?
Un programa multi-hilo es un proceso que lanza varios hilos de ejecución.
Las ventajas de estos programas frente a los mono-hilo es que son mas eficientes respecto a los tiempos de ejecución.
Cuando se tienen varios procesadores o un procesador con varios núcleos la ventaja es evidente, ya que distintos hilos corren simultaneo.
Sin embargo también trae beneficios aunque se trabaje en un sistema con un solo procesador. Esto se debe a que durante el ciclo de vida de un hilo existen tiempos muertos de espera en interfaces de entrada/salida. Tiempos que son aprovechados para darle tiempo de procesador a otro hilo que esté listo para seguir ejecutandose.

¿Qué diferencias existen entre un Proceso y un Hilo?
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:

	Entre Procesos	Entre Hilos
Variables locales no static y automáticas	No	No
Variables globales y locales static	No	Si
Code Segment	No *	Si
Heap	No	Si
File Descirptors (Archivos, Sockets, etc)	No **	Si
Colas I/O	No	No ***
Registros del procesador	No	No

* En algunos sistemas operativos el code segment es el mismo para distintos procesos.
** Cuando un proceso lanza a otro, dependiendo del SO y de los parámetros de lanzamiento, puede heredar sus archivos abiertos.
*** Depende del sistema operativo.

<simil>
¿Qué es un thread? Describa sus características en términos de area de memoria y otras características que le son propias y que comparte con el resto de los threads de un programa.
Un hilo es simplemente una tarea que puede ser ejecutada al mismo tiempo que otra tarea. (Ver Pregunta ¿Qué diferencias existen entre un Proceso y un Hilo?)

¿Qué elementos conoce para coordinar el acceso de distintos threads a recursos compartidos? Explique el uso de uno de ellos.
<simil>
¿Qué recursos conoce para que 2 o más threads controlen el acceso concurrente a estructuras de memoria comunes?

- Un método de sincronización es el de la **exclusión mut**ua (mutex). Se protege un recurso compartido para que no pueda ser accedido por más de un hilo a la vez.
- Otro método de sincronización es el **semaforo**. Simplfica la resolución de ciertos problemas en paralelo, pero tiene el problemas para ser compilador según la plataforma a la que se migre.

(6) ¿Qué es un Mutex?. Metodos disponibles de su uso. Ejemplifique.
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.
Ejemplo de mutex con recurso protegido:
Cuando el primer hilo (A) llegue a la línea 7 encontrará el mutex liberado. Lo tomará inmediatamente y procederá a acceder al recurso en la línea 8. Si en ese instante el hilo B llega a la línea 6 suspenderá su ejecución a la espera de que A libere el mutex (lo hará cuando llegue a la línea 9).

```
1.  #include <mutex>
2.  #include <thread>
3.  std::mutex m; //Mutex global
4.  int recursoCompartido = 0;
5.
6.  void foo(){
7.      m.lock(); // Se solicita el mutex
8.      recursoCompartido++; //Accedo al recurso protegido
9.      m.unlock(); // Se libera el mutex
10. }
11.
12. int main() {
13.     std::thread tA, tB;
14.     tA = std::thread(foo); //Instancio el hilo A que corre en la funcion foo()
15.     tB = std::thread(foo); //Instancio el hilo B que corre en la funcion foo()
16.     tA.join(); //Espera a que el hilo A termine y luego lo une al hilo principal (main)
17.     tB.join(); //Espera a que el hilo B termine y luego lo une al hilo principal (main)
18.     return 0;
19. }
```

Ejemplo de recurso no protegido:

```cpp
#include <iostream>
#include <thread>
#include <mutex>
#define VALOR 0
#define ITERACIONES 10000
#define N_THREADS 2
std::mutex m; //Mutex global
int a = VALOR; //Recurso compartido global

void foo(int tid){
    for (int i=0; i<ITERACIONES; ++i) {
        a++;
        a--;
        m.lock();
        //Cuando haya solapamiento entre threads va a!=VALOR en esta linea
        //Esto va a suceder porque a++ y a-- no estan lockeadas (no protego el recurso compartido)
        if (a != VALOR) {
            std::thread::id realId = std::this_thread::get_id(); //Obtengo id de un thread
            std::cout << "Thread " << tid << " " << realId << " ITERACION " << i << " a = " << a << std::endl;
            a = VALOR; //Reseteo a para normalizarlo
        }
        m.unlock();
    }
}

int main() {
    std::thread t[N_THREADS];
    //Lanzo un grupo de N threads
    for (int i = 1; i <= N_THREADS; ++i)
        t[i] = std::thread(foo, i);
    //Se unen los threads al thread principal (main)
    for (int i = 1; i <= N_THREADS; ++i)
        t[i].join();
    return 0;
}
```

(4) ¿Qué es un Deadlock? Ejemplifique mediante pseudocódigo / genere un DEADLOCK indefectible. / Haciendo uso de columnas escriba el código de 2 threads que indefectiblemente produzcan deadlock.

Un deadlock, tambien conocido como bloqueo mutuo o interbloqueo se produce cuando dos o más procesos que corren en hilos diferentes están esperando un evento que no va a suceder.
Por ejemplo el proceso 1 tiene tomado el recurso A, y se queda esperando que se libere el recurso B para liberar el recurso A y tomar el recurso B.
Al mismo tiempo, el proceso 2 tiene tomado el recurso B, y se queda esperando que se libere el recurso A para liberar el recurso B y luego tomar el recurso A.
De ésta manera ambos procesos se bloquean mutuamente.

Condiciones para el bloqueo.
Estas son condiciones necesarias para que se presente el interbloqueo.

Condición	Descripción
Exclusión Mutua.	Los recursos son de uso exclusivo. sólo un proceso puede hacer uso de un recurso
Retención y Espera	El proceso mantiene la posesión del recurso mientras espera recursos adicionales.
No apropiación	El proceso no suelta el recurso hasta que termine su uso.
Espera Circular.	Los proceso tiene uno o mas recursos que son requeridos por el siguiente proceso.

Si un hilo trata de tomar un mutex que ya tiene tomado puede producir un deadlock.

Ejemplo de deadlock:

```cpp
/* Se generó un deadlock porque ambos hilos quedaron bloqueados a la
 * espera de un evento que nunca va a suceder. */

#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 hilo principal
    //Hace cosas
    m.unlock(); //Nunca llega a liberar el mutex porque se bloqueo 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 bloqueo en el join
    return 0;
}
```

(4) ¿Qué función utiliza para lanzar un nuevo hilo de trabajo (thread)? Describa su uso y sus parámetros. Ejemplifique.
Se utiliza la funcion std::thread()
Un objeto thread inicializado representa un hilo activo en ejecución. Dicho hilo es joineable y tiene un id único.
En cambio, un thread sin inicializar es un objeto no joineable. Todos los threads en esta situación comparten el mismo id.

```
1.    std::thread first (foo);     //Lanza un nuevo hilo que llama a la función foo()
2.    std::thread second (bar,0);  //Lanza un nuevo hilo que llama a la función bar(0)
```

(2) ¿Cómo puede saberse si un thread se encuentra activo o si ya terminó su ejecución?

(3)
Explique el propósito de la función pthread_join. Ejemplifique. (equivalente a thread::join() en c++11)
Su equivalente para el estandar de C++1 es la función join() de <thread>
Bloquea el hilo actual hasta que el hilo identificado por *this (ej: t1.join(), this* es t1) termine su ejecución.
En otras palabras espera a que el hilo al que pedimos unir termine su ejecución para unirlo a nuestro hilo.
No retorna nada.

```
1.    int main() {
2.        std::thread tA, tB;
3.        tA = std::thread(foo); //Instancio el hilo A que corre en la funcion foo()
4.        tB = std::thread(foo); //Instancio el hilo B que corre en la funcion foo()
5.        tA.join(); //Espera a que el hilo A termine y luego lo une al hilo principal (main)
6.        tB.join(); //Espera a que el hilo B termine y luego lo une al hilo principal (main)
7.        return 0;
8.    }
```

<simil>
(2) ¿Qué función se utiliza para esperar la terminación de un thread? Descríbala. Que devuelve?
Supongo que se refiere a join()

¿Puede la función pthread_join lockear algún hilo? Justifique. (equivalente a thread::join() en c++11)
La función thread::join() lockea al hilo (A) que la lanza. Esto se debe a que queda bloqueado a la espera de que el hilo (B) que trata de joinear (unir) se libere.
Si el hilo B se encuentra bloqueado por alguna razón, no permitirá que el hilo A lo joinee bloqueandolo.

Describa y ejemplifique un método utilizado para esperar la terminación de 2 o más threads.
Se puede utilizar pthread_barrier()

```
1.    #include <stdio.h>
2.    #include <stdlib.h>
3.    #include <unistd.h>
4.    #include <pthread.h>
5.    #define THREAD_COUNT 4
6.    pthread_barrier_t mybarrier;
7.
8.    void* callback(void *id_ptr) {
9.      sleep(5);
10.     pthread_barrier_wait(&mybarrier); //Bloquea hasta que a la barrera lleguen los 4 hilos+ el principal.
11.     return NULL;
12.   }
13.
14.   int main() {
15.       pthread_t ids[THREAD_COUNT];
16.       int short_ids[THREAD_COUNT];
17.       pthread_barrier_init(&mybarrier, NULL, THREAD_COUNT+1); //Creo una barrera para 5 hilos (4+el principal).
18.
19.       for (int i = 0; i < THREAD_COUNT; ++i) {//Lanzo 4 hilos en la funcion callback
20.           short_ids[i] = i;
21.           pthread_create(&ids[i], NULL, callback, &short_ids[i]);
22.       }
23.
24.       pthread_barrier_wait(&mybarrier); //Bloquea hasta que a la barrera lleguen los 4 hilos+ el principal.
25.
26.       for (int j = 0; j < THREAD_COUNT; ++j) //Uno todos los hilos en el hilo principal
27.           pthread_join(ids[j], NULL);
28.
29.       pthread_barrier_destroy(&mybarrier); //Destruyo la barrera
30.       return 0;
31.   }
```

Práctica / Programar (6) (0/6)

1) Escriba un ejemplo en el cual un thread incremente un contador global entero de a una unidad; y otro thread del mismo proceso lo imprima. Ambos threads deben coordinar el acceso a la memoria y deben correr sincronizados de forma que siempre se impriman los números 1,2,3,4....

2) Escriba un ejemplo en el cual un thread use un contador global entero y lo incremente de a una unidad; y otro thread del mismo proceso lo imprima y lo decremente. Ambos threads deben coordinar el acceso a la memoria y deben correr sincronizados de forma que siempre se impriman los números 1,0,1,0....

3) Escriba un ejemplo en el cual un thread genere los nros. de la serie de fibonacci a razón de 1 cada 2 segundos; y otro thread del mismo proceso los imprima al mismo ritmo. Ambos threads deben coordinar el acceso a la memoria y deben correr sincronizados de forma que la serie se imprima correctamente.

4) 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 coherencia de la estructura en todo momento.

5) Escriba el pseudocódigo de 2 threads que implementen el concepto de producer-consumer, utilizando una estructura en común y esperándose entre ellos. No se ocupe de la condición de finalización.

```
#include <iostream>
#include <thread>
#include <mutex>
#include <unistd.h>

std::mutex m;
int contador;
bool turno; //false 1er thread, true 2do thread

void fooHilo1() {
    while (true) {
        m.lock();
        if (!turno) {
            contador++;
            std::cout << contador << std::endl;
            turno = true;
            sleep(1);
        }
        m.unlock();
    }
}

void fooHilo2() {
    while (true) {
        m.lock();
        if (turno) {
            contador--;
            std::cout << contador << std::endl;
            turno = false;
            sleep(1);
        }
        m.unlock();
    }
}

int main() {
    contador = 0;
    turno = false;
    std::thread t1(fooHilo1);
    std::thread t2(fooHilo2);
    t1.join();
    t2.join();
    return 0;
}
```

6) Sumponga que está programando una aplicación multi-thread y necesita lanzar un thread pasándole 3 parámetros. ¿Cómo lo hace?. Ejemplifique.

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

void foo(int arg1, char arg2, bool arg3) {
    std::cout << arg1 << " " << arg2 << " " << arg3 << std::endl;
}

int main() {
    std::thread t1(foo, 33, 'c', true);
    t1.join();
    return 0;
}
```

GTK/GTKMM

Cierre ordenado (3) (1/2)

1) **(2)** Escriba una aplicación básica con interfaz gráfica. El programa debe desplegar una ventana con un botón “Cerrar” (que haga terminar ordenadamente la aplicación). Comente/Explique el programa.

2) Escriba un programa para ambiente gráfico Windows/Linux que cree una ventana. Incluya código necesario para hacer el cierre ordenado de la misma.

Combobox / ListBox (5) (2/3)

1) Escriba la función OrdenarCombo que tome el contenido de un listbox/combobox (recibido como parámetro) y lo escriba en el mismo control pero en orden inverso.

2) Escriba una rutina que lea los elementos de un listbox/combobox, los ordene alfabéticamente y los escriba nuevamente en el control.

Solución

```
1.  void ordenar(Gtk::ComboBoxText* combo) {
2.      combo->set_active(0);
3.      Glib::ustring texto;
4.      std::list< Glib::ustring > contenido;
5.      //Capturo los elementos del combobox en una lista
6.      while (combo->get_active_row_number() != -1) {
7.          texto = combo->get_active_text();
8.          combo->remove_text(texto);
9.          contenido.push_back(texto);
10.         combo->set_active(0);
11.     }
12.     //Ordeno la lista
13.     contenido.sort();
14.     //Cargo el combobox nuevamente
15.     std::list< Glib::ustring >::const_iterator it = contenido.begin();
16.     for (contenido.begin(); it != contenido.end(); ++it)
17.         combo->append_text(*it);
18. }
```

3) **(2)** Escriba una rutina que lea los elementos de un listbox/combobox, los pase a minúscula/mayuscula y los escriba nuevamente en el control.

Solución

```
1.  void mayuscula(Gtk::ComboBoxText* combo) {
2.      combo->set_active(0);
3.      Glib::ustring texto;
4.      std::list< Glib::ustring > contenido; //Creo un vector de strings
5.      while (combo->get_active_row_number() != -1) { //Recorro todos los elementos
6.          texto = combo->get_active_text(); //Seleccionas el texto activo
7.          combo->remove_text(texto); //Borras el elemento
8.          contenido.push_back(texto.uppercase()); //Lo pasas a mayuscula y lo guardas en el vector, lowercase() para pasar a minuscula
9.          combo->set_active(0); //Selecciona el elemento 0
10.     }
11.     std::list< Glib::ustring >::const_iterator it = contenido.begin(); //Interador a traves de un vector
12.     for (contenido.begin(); it != contenido.end(); ++it) //Recorre el vector
13.         combo->append(*it); //Cargo de nuevo los elementos del vector en el combobox
14. }
```

(5)

4) **(3)** Escriba el trozo de código necesario para cargar un Combobox/Listbox de una ventana con 20 strings del tipo “Opción xx”, siendo x un entero entre 1 y 20.

Solución

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

int main(int argc, char* argv[]) {
    Gtk::Main kit(argc, argv);
    Gtk::ComboBoxText combo;

    Glib::ustring str = "Opcion ";
    std::stringstream sstream;
    for (int i = 1; i <= 20; ++i) {
        sstream << i;
        combo.append_text(str + sstream.str());
        sstream.str("");
    }

    Gtk::Window v;
    v.add(combo);
    v.show_all();
    Gtk::Main::run(v);
    return 0;
}
```

<simil>

5) **(2)** Escriba la función CargarCombo que complete el Combobox/Listbox pasado como parámetro con 10 strings de la forma “Opción xx', siendo xx '01','02',...'10'.

Escriba el trozo de código necesario para obtener el texto de la opción seleccionada en un combo de una ventana.

```
Glib::ustring string = combo.get_active_text(); //donde combo es de tipo ComboBoxText.
```

Editbox (9) (4/6)

1) **(3)** Escriba la función MAYUSCULA que tome el contenido de un Edit de una ventana (recibido como parámetro) y lo escriba en el mismo control poniéndo en mayúscula.

2) Escriba la función FORMATEAR que tome el contenido de un Edit de una ventana (recibido como parámetro) y lo escriba en el mismo control poniendo la primera letra en mayuscula.

3) Escriba un trozo de código que: tome el contenido de un EDIT, le saque los espacios y ponga el resultado como contenido de ese mismo EDIT.

4) Escriba una rutina que le de formato al contenido de un edit. El formato deseado consiste en pasar a mayúscula la primer letra de cada palabra y dejar el resto de los caracteres en minúscula.

5) Escriba la función FORMATEAR que tome el contenido de un Edit de una ventana (recibido como parámetro) y lo escriba en el mismo control con 2 decimales. En el caso que el contenido leído no sea numérico, debe escribir “***ERR***”

(2)

Escriba la función FORMATEAR_FECHA que le de formato al texto de un campo de texto(edit) de una ventana. Esta función deberá leer la fecha escrita en el control (con barras y/o puntos de separación) y reescribirla con el formato dd/mm/aaaa. Por ejemplo: '01012010' se transformará en '01/01/2010'; '1.1.2010' en '01/01/2010' y '01/01/2010' en '01/01/2010'. Cabe destacar que el control será recibido como parámetro de la función.

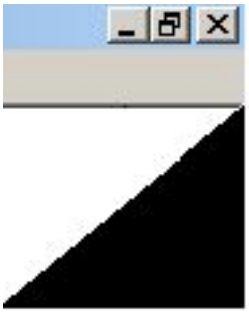
<simil>

Escriba la función FORMATEAR_PRECIO que le de formato al texto de un campo de texto(edit) de una ventana. Esta función deberá leer un número escrito en el control (con signo $ y/o punto decimal) y reescribirlo con el formato $ ddddd.dd. Por ejemplo: '$10' se transformará en '$ 10.00'; '10.05' en '$ 10.05' y '$10.3' en '$ 10.30'. Cabe destacar que el control será recibido como parámetro de la función.

Dibujar (12) (6/9)

Describa el mecanismo que se utiliza crear una ventana con un area de dibujo.
Para crear una ventana con un area de trabajo no se hace mas que crear una ventana y se le agregar el drawing area
Si queremos que se dibuje algo, hacemos una clase que herede de DrawingArea y redefinimos on_expose_event

1) Escriba una rutina (para ambiente gráfico Windows o Linux) que dibuje la siguiente figura:

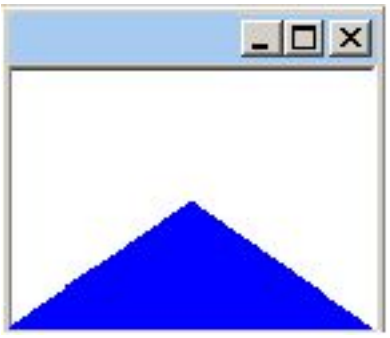


(3)
2) Escriba una rutina (para ambiente gráfico Windows o Linux) que dibuje la siguiente imagen en su área de dibujo.

<simil>
Escriba una rutina gráfica (Linux o Windows) para dibujar un óvalo que abarque todo el área de trabajo de una ventana.
<simil>
Implemente una rutina (en Windows o Linux) que dibuje un óvalo que ocupe toda la ventana.

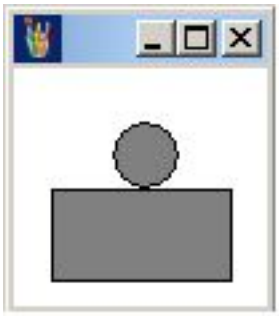
3) Escriba una rutina gráfica que realice el siguiente dibujo:

4) Escriba una rutina (para ambiente gráfico Windows o Linux) que pinte un triángulo azul con las carcterísticas que muestra la siguiente figura:



5) **(2)** Escriba una rutina gráfica que realice el siguiente dibujo en la ventana:

6) Escriba una rutina gráfica para Windows o Linux que dibuje lo que muestra la siguiente figura:



Escriba una rutina gráfica que dibuje una línea diagonal descendente que ocupe toda la pantalla.

Escriba una rutina gráfica que pinte la pantalla de color azul.

Escriba una rutina (para ambiente gráfico Windows o Linux) que dibuje 2 lineas diagonales dividiendo la ventana en 3 franjas de igual ancho.

Manejo de Archivos (25) (3/12)

El segundo parámetro de la función fopen indica el modo de apertura del archivo. ¿Cuales son los modos posibles?

"r" Abre un archivo para lectura. Si no existe devuelve error.
"w" Abre un archivo para escritura. Si existe, borra el contenido y luego escribe. Si no existe lo crea y luego escribe.
"a" Abre un archivo para agregar datos. Agrega información al final del archivo (fseek, fsetpos, rewind) estan deshabilitadas. Crea el archivo si no existe.
"r+" Abre un archivo para lectura/escritura. Si no existe devuelve error.
"w+" Abre un archivo para lectura/escritura. Si existe, borra el contenido y luego lee/escribe. Si no existe lo crea y luego lee/escribe.
"a+" Abre un archivo para agregar datos o lectura. Agrega información al final del archivo (fseek, fsetpos, rewind) estan deshabilitadas. Crea el archivo si no existe.

Al modo de apertura se le puede indicar explicitamente el tipo de traducción a utilizar. Si no se indica nada por default es para archivos de texto.

- t (archivo de texto Ej: rt
- b (archivo binario) Ej: rb
- implicito (archivo de texto) Ej: r

Práctica

!feof(fdR); //Si usamos la función feof tener en cuenta que corta cuando se hace un intento de lectura y ya se termino el archivo.
Considerar ésto ya que sino podemos caer en entrar al while una vez más (Es decir, entra una vez demás).
Una solución es hacer un get antes de entrar al while y despues hacer siempre el get al final para dentro del while ya estar pidiendo el siguiente, y si es el EOF corte.
Otra solucion es usar (fread() == de a la cantidad que leemos), si lee menos es porque llegó al fin de archivo, y encima estaba corrupta la multiplicidad del archivo.

```
FILE* fopen(FILENAME, FORMAT);
fclose(FILE* fd);
fread(dirBuffer, tamañoElemento, cantidadElementos, FILE* fd); //Lee del fd la cantidad de bytes "cantidadElementos*tamañoElemento" y guarda en el buffer.
fwrite(dirBuffer, tamañoElemento, cantidadElementos, FILE* fd); //Lee del buffer la cantidad de bytes "cantidadElementos*tamañoElemento" y lo escribe en el fd.
bool feof(FILE* fd) //Devuelve true si se encontró el caracter EOF (si en la ultima lectura se encontro un EOF).
fseek(FILE* fd, int offset, int origen); //Mueve el puntero fd al origen+offset
Constantes:
SEEK_SET  Beginning of file
SEEK_CUR  Current position of the file pointer
SEEK_END  End of file *
```

Para resolver ejercicios en los que nos piden editar un archivo sin crear archivos intermedios el truco está en utilizar dos punteros. Uno para lectura (fdR) y otro para escritura (fdW).
Tipos de ejercicios:
Caso 1: Se **acorta** el tamaño del archivo porque nos piden eliminar información.
Si nos piden eliminar elementos del archivo muy posiblemente el puntero fdR siempre queda por delante del fdW, por lo que a medida que leemos el archivo con el puntero fdR podemos ir escribiendo con el fdW sin riesgo de pisar información útil.
Finalmente truncamos el archivo hasta donde haya llegado el puntero fdW. Es decir con la función: ftruncate(fileno(fdW), ftell(fdW));

Caso 2: Se **agranda** el tamaño del archivo porque nos piden agregar información (por ejemplo duplicar palabras).
Como el fdW avanza rápido que el fdR, lo que tenemos que hacer es una primera pasada para calcular el tamaño original y el offset (lo que se expandio).
Luego debemos agrandar el archivo a tamañoOriginal+offset y desplazar en offset todos los bytes originales (empezando de los últimos a los primeros)
Cerramos los fileDescriptors de lectura y escritura para guardar los cambios de desplazamiento.
Luego procedemos como en el Caso 1 (colocanto el fdR en offset). De esta manera el fdR siempre va a quedar adelante del fdW, ya que inicia adelantado. Ya no hay riesgo de pisar bytes.

(4)
2) Escribir un programa C que, sin crear archivos intermedios, procese el archivo “enteros.dat”. El procesamiento consiste en leer grupos de 2 enteros sin signo de 32 bits desde el archivo y escribir, en su lugar, la resta del grupo (el archivo se **acortará**).
<simil>
3) Desarrolle un programa C que, sin crear archivos intermedios, procese el archivo “datos.bin”. El procesamiento consiste en leer 5 enteros sin signo de 16 bits desde el archivo y reemplazarlos por el promedio del grupo (el archivo se **acortará**).
<simil>
4) Escriba un programa ANSI C que modifique el archivo datos.bin sobre si mismo. El procesamiento consiste en leer 2 enteros de 32 bits y reemplazarlos con un único entero que resulte de hacer la operación OR entre los nros leídos. (el archivo se **acortará**).
<simil>
5) Escribir un programa ANSI C que, sin crear archivos intermedios, procese el archivo “nros.bin”. El procesamiento consiste en leer 2 enteros con signo de 32 bits desde el archivo y escribir, en su lugar, el diferencia en valor absoluto (sin signo), la suma y el número resultante de hacer OR a nivel bits entre ambos números (el archivo se **agrandará**).
<simil>
6) Escribir un programa C que, sin crear archivos intermedios, procese el archivo “datos.int”. El procesamiento consiste en leer 3 enteros sin signo de 32 bits desde el archivo y escribir, en su lugar, el promedio del grupo redondeado siempre para arriba (el archivo se **acortará**).

7) Escribir un programa ISO C que, sin crear archivos intermedios, altere el archivo “data.bin” reemplazando todas las secuencias de 3 bytes 0x34 0x43 0x44 por la secuencia de 2 bytes 0x34 0x43. Cabe destacar que el programa debe reprocesar el reemplazo efectuado. (Ejemplo: 0x34 0x43 0x44 0x44 ---> 0x34 0x43 0x44 ---> 0x34 0x43). (el archivo se **acortará**).
<simil>
Escriba un programa ANSI C que procese el archivo datos.bin. El procesamiento consiste en reemplazar las secuencias de bytes 0x21,0x43,0xA2 por 0x11,0x14. (el archivo se **acortará**).

8) Escribir un algoritmo ANSI C que, sin crear archivos intermedios altere el archivo a.txt reemplazando la secuencia '//' por '*'. Excepto si se encuentra entre parentesis. (el archivo se **acortará**).

(3)
9) Escriba una aplicación C que modifique el archivo a.dat eliminando los bytes mayores a 0xC3. (el archivo se **acortará**).

<simil>
10) **(2)** Escriba una aplicación C que modifique el archivo a.bin sobre sí mismo, eliminado los bytes en posiciones múltiplo de 178. (el archivo se **acortará**).

(4)
Escribir un programa que procese el archivo in.txt sobre sí mismo. El proceso consiste en eliminar las palabras que no tengan más de 3 vocales distintas. (el archivo se **acortará**).
<simil>
Escribir un programa que procese el archivo in.txt sobre sí mismo. El proceso consiste en eliminar las palabras que tengan constantes y números. (el archivo se **acortará**).
<simil>
Escribir un programa C que procese el archivo "fuente.c" sobre sí mismo (sin crear archivos intermedios). El proceso consiste en eliminar todos los comentarios (/*...*/). (el archivo se **acortará**).
<simil>
Escriba un programa C que procese el archivo a.bin sobre sí mismo. El procesamiento consiste en imprimir y eliminar del archivo todas las secuencias del tipo 'ddxxxx' siendo 'd' un dígito y 'x' una letra Mayúscula o minúscula. (el archivo se **acortará**).

(3)
1) Escribir un algoritmo que procese el archivo x.dat invirtiendo los bytes xy por yx. (el archivo **mantiene tamaño**).
<simil>
11) Escribir un programa ISO C que lea el archivo "a.txt" e invierta sus caracteres sin utilizar archivos intermedios. (el archivo **mantiene tamaño**).
<simil>
12) Escriba un programa C que procese el archivo a.bin sobre sí mismo. El procesamiento consiste en invertir el orden de los bytes del archivo en grupos de 3. Ejemplo: Si el archivo original es: abc def ghi ... xyz, el resultado debe ser: cba fed ihg...zyx (el archivo **mantiene tamaño**).
<simil>
Escriba el programa INVERTIR que procese un archivo sobre símismo, invirtiendo el stream de bits del archivo. Por ejemplo, si el archivo contiene los bytes "AA 00 F0 5A", al finalizar la ejecución debe contener "5A 0F 00 55". (el archivo **mantiene tamaño**).
$AA\ 00\ F0\ 5A_{16}$ -> $01011010\ 00001111\ 00000000\ 01010101_2$
$5A\ 0F\ 00\ 55_{16}$ -> $10101010\ 00000000\ 11110000\ 01011010_2$
https://stackoverflow.com/questions/2602823/in-c-c-whats-the-simplest-way-to-reverse-the-order-of-bits-in-a-byte
http://www.geeksforgeeks.org/write-an-efficient-c-program-to-reverse-bits-of-a-number/
https://medium.com/square-corner-blog/reversing-bits-in-c-48a772dc02d7

(3)
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 2 consonantes. (el archivo se **agrandará**)
<simil>
Escribir un programa C que procese el archivo texto.txt sobre sí mismo. El proceso consiste en **triplicar** las palabras que tengan todas las vocales. (el archivo se **agrandará**)
<simil>
Escribir un programa ANSI C que procese el archivo in.txt sobre sí mismo. El proceso consiste en duplicar las palabras que tengan más de 4 vocales distintas. (el archivo se **agrandará**)

(2)
<simil>
Escriba una aplicación C que modifique el archivo a.bin triplicando (repitiendo 3 veces) los bytes en posiciones múltiplo de 172. (el archivo se **agrandará**)
<simil>
Escriba una aplicación C que modifique el archivo a.bin triplicando (repitiendo 3 veces) los bytes mayores a 0xD1. (el archivo se **agrandará**)

Escriba un programa ISO C que procese el archivo de números datos.txt sobre sí mismo. El procesamiento consite en **convertir** los números encontrados (de 1 o más cifras **decimales) a octal**. (el archivo se **agrandará**).

Escriba un programa C que procese el archivo a.bin sobre sí mismo. El procesamiento consiste en incrementar cada entero (2 bytes) del archivo en un número igual a la posición que dicho entero ocupa en el archivo.

Escribir un programa que procese un archivo (binario) de enteros sin signo sobre sí mismo. El procesamiento consiste en leer pares de enteros y reemplazarlos por 3 enteros: su suma, su resta y su división entera. (el archivo se **agrandará**).

(2) Escriba una función C que desplace el contenido de un archivo binario, desde la posición actual, hacia adelante o hacia atrás, sin crear archivos intermedios. La función debe recibir como parámetros un descriptor de archivo y un offset (entero largo) que indique el desplazamiento. Si el contenio debe desplazarse hacia atrás (offset>0), el archivo incrementará su tamaño, quedando los primeros bytes intactos. Por el contrario, si el contenido debe desplazarse hacia adelante (offset<0), los primeros bytes se perderán y serán los últimos los que queden intactos.

Herencia / Polimorfismo (18) (3/8)

(5) ¿Qué es la **herencia**?¿Para qué se utiliza?¿Qué **tipos** conoce?
<simil>
Resuma las accesibilidad de 3 tipos de componentes de una clase (público, protegido y privado) y cómo se modifican en las clases heredadas con los distintos **tipos de herencia**.
<simil>
(3) Explique las **características y usos** de los distintos **tipos de herencia,** haciendo referencia al **acceso permitido y/o denegado** desde **clases dependientes** y desde **fuera de la clase**.
Sirve para crear nuevas clases partiendo de una clase o de una jerarquia de clases preexistente evitando el rediseño, la modificacion y verificacion de la parte ya implementada.
Facilita la creacion de objetos de otros ya existentes e implica que una subclase obtiene todo el comportamiento y eventualmente los atributos de la super clase.

Es una relación dentro de la programación orientada a objetos en donde una clase deriva de otra.
Para especificar o generalizar, para reutilizar la interfaz.
Herencia simple y múltiple (depende de la cantidad clases que se heredan).

Existen 3 tipos de herencia. Publica, protegida y privada.
Pública: los atributos/métodos publicos y protected se mantienen con la misma visibilidad que los de su clase padre.
Protected: los atributos/métodos publicos y protected de la clase padre pasan a ser protegidos en la propia clase (clase hijo). Entonces el único que sabe que hereda del padre es la propia clase (el hijo) y las que hereden de ésta.
Privada: los atributos/métodos publicos 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 la propia clase (el hijo). Las clases que heredan de la clase hija ya no ven los atributos/métodos de la clase padre.

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

Haciendo uso de **herencia**, de un ejemplo práctico de una **clase abstracta**.
La clase abstracta no puede instanciar objetos. Una clase es considerada abstracta si posee al menos un método virtual puro. Por lo tanto las subclases deben obligatoriamente implementar ese método virtual puro.
Un método virtual puro es un método que no está implementado en la super clase, pero los hijos si o si lo deben implementar.

```
1.  class Animal { //Clase Abstracta
2.  public:
3.      virtual void comer() = 0; //Método virtual puro
4.      virtual ~Animal();
5.  };
6.
7.  class Perro : public Animal {
8.  public:
9.      void comer() { //Método virtual puro implementado
10.     ~Perro();
11. };
```

(2) ¿Qué es la **herencia múltiple**? **Escriba declaraciones** de métodos/clases que ejemplifiquen sus conceptos.
La herencia múltiple le permite a una clase heredar los comportamientos (métodos) y las características (atributos) de más de una super clase.

¿Qué recomendación especial tendría Ud. al momento de escribir una **clase que usa memoria dinámica** y que es usada con **polimorfismo**? Ejemplifique.
Tendría la precaución de hacer que el destructor de la clase padre sea un método virtual.
Cuando se instancia b1 en memoria dinamica con el new, el objeto es de tipo B, pero el puntero es de tipo A. Por lo tanto, cuando se realice el delete b1 se llamará al destructor de la clase A ya que vemos al objeto como de tipo A.
Si el ~A() no es virtual entonces nunca se llamará al ~B(), con lo cual no se liberará la memoria asociada a la clase B.
Si el destructor de A es virtual (virtual ~A()) entonces primero se llama al destructor de la clase derivada, es decir a ~B() liberando los recursos de B, y luego de ejecutarse el destructor de B se ejecuta el destructor de A.
Caso delete b2:
Como el puntero es de tipo B, se llama directamenteal destructor de B el cual a su vez llamará luego al destructor de la clase padre A.

Ejemplo:

```
1.  class A {
2.  protected :
3.      int x;
4.  public :
5.      int y;
6.  public :
7.      virtual ~A() {}
8.  };
9.
10. class B : public A {
11. private :
12.     int z;
13. public :
14.     ~B() {}
15. };
16.
17. int main() {
18.     A* b1 = new B(); // Instancio b1 de tipo B, pero el puntero es de tipo A (padre de B).
19.     delete b1; //Llama a ~A(), ve que es virtual -> llama al destructor de la derivada ~B() y luego llama al destructor del padre ~A().
20.     B* b2 = new B(); // Instancio b2 de tipo B, y el puntero es de tipo B.
21.     delete b2; //Llama a ~B() y luego llama al destructor del padre ~A().
22.     return 0;
23. }
```

(2) ¿Qué es el **Polimorfismo**? **Ejemplifique**.

El polimorfismo es la posibilidad de enviar mensajes sintacticamente iguales independientemente del tipo de objeto. El único requisito que deben cumplir los objetos es saber responder el mensaje.
Se resuelve en tiempo de ejecución en función a la clase que pertenece el objeto.

¿Puede existir polimorfismo sin punteros?

Puede, me falta dar ejemplo.

Haciendo uso de **clases** y **métodos virtuales**, de un ejemplo de **polimorfismo**.

Supongamos que la clase Hijo hereda de Padre; que Padre posee varios métodos públicos de utilidad en Hijo, a excepción de 1 que deseamos “ocultar”. ¿Cómo podemos realizar esto? Ejemplifique.

Pregunta de origen dudoso.
http://stackoverflow.com/questions/2141188/changing-function-access-mode-in-derived-class

StandardTemplateLibrary (STL) / Templates (25)(5/14)

(5) ¿Qué es **STL** (Standard Template Library)? **Describa las ventajas que ofrece** a un programador.

Las librerias STL (Standard Template Library) son las librerias de implementaciones standard de objetos templarizados (algoritmos, iteradores, funciones y contenedores).
La misma logra sus objetos utilizando templates y brinda un enfoque en tiempo de compilación.
En STL destacan los contenedores, los cuales se pueden utilizar con cualquier tipo de dato que admita algunas operaciones elementales (como la copia y la asignación).
Ventajas:

- Es estándar, por lo que está disponible en todos los compiladores y plataformas. Esto permite utilizar la misma librería en todos los proyectos.
- Soporta tipos de datos creados por el usuario siempre y cuando se sobrecarguen los operadores necesarios.
- Permite abstraerse de las implementaciones de distintas estructuras de contenedores, y permite recorrerlos a través de iteradores de forma estandarizada.
- Permite al programador aprovechar la eficiencia de los algoritmos implementados en la librería.

<simil>

¿Qué **ventajas y desventajas** ofrecen las librerías **STL** frente a una implementación **con punteros** del correspondiente **TDA**?

Ventajas: permite evitar el uso de casteos manuales para adaptar las distintas estructuras a un TDA estandar.

<simil>

Para implementar una “Lista Genérica” se discuten 2 enfoques: a) Uso de Templates; b) Implementación de una lista de void *. ¿Qué ventajas ofrece cada una de ellas?

Los punteros a void se utilizan para permitir a funciones operar con tipos de datos desconocidos.
Desventajas de void*:

- El compilador no puede hacer chequeo de tipos (ya que no los distingue).
- Al no conocer el tipo no puede llamar a los operadores sobrecargados de dicho tipo.
- Para utilizarlo se deben realizar muchos casteos (lo cual es peligroso).

Ventajas de Templates:

- Permiten mejorar la claridad del código
- Se realiza chequeo de tipos en tiempo de compilación.
- Se evitan los casteos.

Desventajas de Templates:

- Puede crecer mucho el ejecutable de forma innecesaria porque hace copy paste de código para generar nuevas clases (Code bloat).
- El programa demora más en ser compilado.

¿Qué son los **templates?**. **Explique y Ejemplifique**.

La programación genérica es un tipo de programación que está mucho más centrada en los algoritmos que en los datos.
Es decir que en la medida de lo posible, los algoritmos deben ser parametrizados al máximo, permitiendo así que puedan servir para la mayor variedad posible de tipos y estructuras de datos. A raiz de ello nace lo que se conoce como Templates/Plantillas.
Por ejemplo, si quisieramos implementar una clase Lista, deberíamos implementar una clase Lista distinta por cada tipo de dato que contenga. Pero si miramos la implementación, lo único que cambia entre las distintas clases Lista que tenemos es el tipo de dato con el que trabajan, porque la esencia de los algoritmos es la misma. Utilizando una Plantilla Lista podemos abstraernos del tipo de dato para centrarnos en la implementación genérica, y luego eventualmente realizar algunas especializaciones para mejorar le eficiencia cuando se trabaje con tipos de datos concretos.

¿Por qué motivo las librerías STL son distribuidas mediante su Código Fuente y no compiladas en una .lib/.obj/.o?

Las librerías STL son distribuidas mediante código fuente porque las mismas utilizan templates/plantillas y el código objeto se debe generar en funciónde los tipos que utilicemos en nuestro código.
Es decir, se realiza un copy paste de la plantilla para el tipo genérico T y se reemplaza el tipo de dato que utilicemos y luego se genera código objeto a raiz del código fuente generado con la plantilla.

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

(2) ¿Qué es un **iterador** de **STL**? ¿Cómo se utiliza? **Ejemplifique**.

Un iterador de STL es un objeto de alguna clase contenedora de elementos templarizados.
Sirve para recorrer una estructura contenedora y acceder a sus elementos sin necesidad de conocer la implementación interna de dicha estructura.
Usualmente se lo utiliza para busquedas, consultas de los elementos, etc.
Ejemplo:

```
1.  #include <iostream>
2.  #include <list>
3.  int main() {
4.      std::list<int> ls = {1, 2, 3}; //Creo con un template una lista de ints (enteros)
5.      std::list<int>::iterator it; //Creo un iterador para recorrer list<int>
6.      for (it=ls.begin(); it!=ls.end(); ++it) //Recorro la lista utilizando el iterador
7.          std::cout << *it << std::endl; //Imprimo valores
8.      return 0;
9.  }
```

¿Qué **clase** ofrece el paquete **STL** para encapsular una **lista? Ejemplifique** su uso.

El paquete STL ofrece la clase std::list<>
Ejemplo:

```
10.  #include <iostream>
11.  #include <list>
12.  int main() {
13.      std::list<int> ls = {1, 2, 3}; //Creo con un template una lista de ints (enteros)
14.      std::list<int>::iterator it; //Creo un iterador para recorrer list<int>
15.      for (it=ls.begin(); it!=ls.end(); ++it) //Recorro la lista utilizando el iterador
16.          std::cout << *it << std::endl; //Imprimo valores
17.      return 0;
18.  }
```

Usando templates, escriba una rutina que cargue una lista STL de objetos con información ingresada por teclado. Asuma que los objetos de la lista poseen el operador >> para inicializarse desde cin.

Haciendo uso de STL defina una clase MATRIZ, de forma que las siguientes operaciones sean válidas:

TMatriz<3,3> Matriz;
float F;
Matriz[1][2]=3.4;
F=Matriz[1][2];

Haciendo uso de listas STL, declare una matriz genérica de tipo T. Escriba una nota donde aclare qué operadores asume que existirán en T.

Utilizando templates defina las clases necesarias para que el siguiente código sea válido:
Arreglo<int> A(5000);
A[0]=123;
A[1]=A[0];

Escriba un rutina C++ que cargue un conjunto de strings a una lista STL. La carga debe finalizar cuando se ingrese el string “QUIT” (que no debe incluirse en la lista).

Escriba una función C++ que ordene el contenido de una lista STL de cadenas alfabéticamente.

Escriba una función que reciba por referencia una lista genérica de tipos X. La rutina debe leer cadenas de la entrada estándar, asumiendo que la clase X posee un constructor con cadena de caracteres, crear objetos del tipo X y cargarlos a la lista. La carga debe terminar cuando se ingrese un string de la forma hexadecimal <0D><0A><0D><0A>.

(2)
1) Defina el operador>> de forma que cargue el contenido de una lista de STL. La carga se iniciará ingresando el número de elementos a cargar.
El paquete STL ofrece la clase std::list<>

```
1.  std::istream& operator>>(std::istream& is, std::list<int>& lista) {
2.      std::string entry;
3.      //Paso el contenido de is (capturado del cin) a entry
4.      std::getline(is, entry);
5.      //Convierto el string entry a char* y luego a int para meterlo en la lista
6.      lista.push_back(atoi(entry.c_str()));
7.      return is;
8.  }
```

<simil>
Defina el operador>> de forma que cargue, desde stdin, una lista STL de strings. La carga debe terminar cuando se ingrese una cadena nula.

(5)
Defina el operador int que trabaje con listas STL de tipos genéricos T. El operador debe devolver la cantidad de elementos distintos que posee la lista.
El enunciado probablemente esté mal armando. Ya que no se puede sobrecargar un operador de conversion (de casteo) de manera global. Y de forma local no podemos hacerlo porque no tenemos acceso a la clase de STL de la librería.

Si tenemos acceso local al STD, entonces la respuesta sería declarar dentro de STD

```
std::operator int (const std::list<T>& ls) {return ls.size();}
```

<simil>
(2) Defina el operador global- que opere con 2 listas STL de tipos genérico T. La función debe devolver una lista con los elementos de la primera que no están en la segunda. Escriba una nota donde aclare que operadores asume que existirán en T.
<simil>
Defina el operador global- que trabaje con listas STL de tipos genéricos T. El operador debe eliminarle a la primer cadena la primera instancia de los elementos existentes en la segunda. Escriba una nota donde aclare que operadores asume que existirán en T.
<simil>
3) Defina el operador global + para obtener la unión de 2 listas STL de tipos genérico T (elementos de ambas listas, sin repeticiones).

Definiciones / Declaraciones (52) (5/6)

Declarar: es decirle al compilador el nombre y el tipo de la variable, o el nombre de la funciòn, su retorno y tipos de argumentos, para luego ser definida posteriormente.
Definir: el compilador aloca memoria para esa variable/función y posiblemente inicialice su contenido.

Punteros a función: http://www.zator.com/Cpp/E4_2_4.htm

Declaración: Asocia un identificador con un tipo de dato (existencia semántica). Se puede declarar múltiples veces.
Variable:
```
extern int x; //Sólo declara la variable, la definición se realiza en otro archivo.
```
Función: (se llama prototipo a la declaración de una función)
```
int suma(int a, int b);
```
Definicion: Asocia un identificador con un tipo de dato y le asigna espacio en memoria. Solo se puede definir una vez.
Variable:
```
int x; //Declara y define la variable (ya que le asigna espacio en memoria).
```
Función:
```
int suma(int a, int b) {
   return a+b;
}
```
Inicialización: Asignar valores concretos al objeto.
Variable:
```
int x; //Se declara y define la variable.
x = 3; //Se inicializa.
```

C/C++
Variable Extern Global
Declara la variable sin definirla. Le indica al compilador que la variable se encuentra definida en otro archivo (para no reasignar otra vez memoria), la necesitamos declarada en el archivo para la asociación semántica y así compile el programa.
Variable Static Global
Es una variable con visibilidad únicamente dentro de ese archivo.
Variable Static Local
La visibilidad de la variable sigue siendo local, pero su valor permanece luego de salir del scope de la función. Cuando la función vuelva a ejecutarse la misma tendrá el valor anterior. Es decir que va a permanecer en su celda de memoria durante toda la ejecución del programa pudiendo ser accedida por punteros ya que se aloja en el segmento de datos y es inicializada una única vez.
Funcion Static
Es una función con visibilidad únicamente dentro de ese archivo.
Funcion Extern
Cuando declaramos una función (el prototipo) en general lo hacemos sin usar extern. Pero por default, si no ponemos nada el compilador sabe que es extern (es decir que solo se la está declarando).

C++
Atributo Static Local
Todas las instancias de una misma clase van a compartir el mismo atributo con su respectivo valor.

Donde se almacenan las variables?

Variables	Almacenadas en
auto	Stack
global	Data Segment
static	Data Segment
register	Registro de la CPU si es posible, sino en el Stack
extern	no especifica donde se almacena la variable, la variable se almacena donde el almacenador lo especifica
const	no especifica donde se almacena la variable, la variable se almacena donde el almacenador lo especifica
volatile	no especifica donde se almacena la variable, la variable se almacena donde el almacenador lo especifica

Almacenamiento lógico

Code Segment [.text]
Es de tamaño fijo y de sólo lectura.
Almacena todas las instrucciones en código máquina que componen el programa.
Existen casos en que varias instancias de programas en ejecución comparten un Code Segment (ej: librerías dinámicas DLLS). Se hace por un tema de eficiencia y espacio. No hay peligro de corrupción ya que cada instancia de programa posee su propio segmento de datos.
Data Segment [.data]
Es de tamaño fijo y permite la lectura/escritura.
Almacena las variables globales inicializadas del programa. Las variables static.
BSS Segment [.bss]
Es de tamaño fijo y permite la lectura/escritura.
Almacena las variables globales sin inicializar.
Heap Segment (Montículo)
Es de tamaño variable y permite la lectura/escritura. Crece a medida que se reserva memoria malloc(), calloc(), o realloc(), new y decrece cuando se libera free(). delete.
Almacena las variables que se almacenen en memoria dinámica.
Stack Segment (Pila)
Es de tamaño variable y permite la lectura/escritura.
Almacena los argumentos pasados al programa, cadenas de retorno, argumentos pasados a las funciones, variables locales (que no sean static), variables auto, valores de retorno de una funcion.

Describa las siguientes declaraciones/definiciones globales:

```
1.  extern   char *a;
2.  static   char a;
3.           char a(char *a) {return *a;}
4.  extern   int (*B); //Equivalente a extern int* B
5.  static   signed int C;
6.  static   int *I[3];
7.  extern   signed int (*A)[1];
8.  extern   short (*S)[2];
9.  extern   short int S[2];
10. static   long int L[2];
11. extern   double (*D)[3];
12. static   double (*I)[3];
13. static   float resta(float a, float b) {return a-b;}
14. static   int resta(float a, double b) {return a-b;}
15.          float calcular(char a, double **d);
16. static   float *A[3];
17. extern   double (*B)[1];
18. static   float RESTA(float a, float b) {return a-b;}
19.          float *a;
20. static   float* X(float *a) {return a;}
21. extern   float *a;
22. extern   char (*X)[2];
23.          float Z(int *Y[5]);
24. static   long Z;
```

1. Se declara la variable 'a' que es un puntero a char definida en un archivo externo.
2. Se declara y define la variable 'a' que es de tipo char sólo visible dentro del archivo donde es declarada.
3. Se declara la función 'a' que recibe como argumento 'a' un puntero a char y retorna un char. Y se la define retornando el parámetro 'a' desreferenciado (retorna lo que apunta 'a').
4. Se declara la variable 'B' que apunta a un int definida en un archivo externo.
5. Se declara y define la variable 'C' que es de tipo signed int (entero signado) sólo visible dentro del archivo donde es declarada.
6. Se declara y define la variable 'I' que es un array de tres posiciones de punteros a int (entero) sólo visible dentro del archivo donde es declarada.
7. Se declara la variable 'A' que apunta a un array de una posicion de tipo signed int (entero signado) definida en un archivo externo.
8. Se declara la variable 'S' que apunta a un array de dos posiciones de tipo short int (enteros cortos) definida en un archivo externo.
9. Se declara la variable 'S' que es un array de dos posiciones de tipo shot int (entero corto) que se encuentra definida en un archivo externo.
10. Se declara y define la variable 'L' que es un array de dos posiciones de tipo long int (entero largo) sólo visible dentro del archivo donde es declarada.
11. Se declara la variable 'D' que apunta a un array de tres posiciones de tipo double (flotante de doble precisión) definida en un archivo externo.
12. Se declara y define la variable 'I' que apunta a un array de tres posiciones de tipo double (flotante de doble precisión) sólo visible dentro del archivo donde es declarada.
13. Se declara la función 'resta' que recibe como primer argumento a 'a' de tipo float (flotante) y como segundo argumento a 'b' de tipo float (flotante) retornando un float (flotante) y sólo visible en el archivo donde es declarada. Y se la define retornando la resta de los argumentos a - b.
14. Se declara la función 'resta' que recibe como primer argumento a 'a' de tipo float (flotante y como segundo argumento a 'b' de tipo double (flotante de doble precisión) retornando un int (entero) y sólo visible dentro del archivo donde es declarada. Se la define retornando la resta de los argumentos a - b.
15. Se declara la función calcular que recibe como primer argumento a 'a' de tipo char y como segundo argumento 'd' un puntero doble a double (flotante de doble precisión) retornando un float (flotante).
16. Se declara y define la variable 'A' qie es im array de tres posiciones de puntero a float (flotante) sólo visible desde el archivo donde es declarada.
17. Se declara la variable 'B' que apunta a un array de una posición de tipo double (flotante de doble precisión) definida en un archivo externo.
18. Se declara la funcion 'RESTA' que recibe como primer argumento a 'a' de tipo float (flotante) y como segundo argumento a 'b' de tipo float (flotante) retornando un float (flotante) y sólo visible desde el archivo donde es declarada. Y se la define retornando la resta de los argumentos a - b.
19. Se declara y define la variable 'a' que es un puntero a floar (flotante).
20. Se declara la función X que recibe como argumento a 'a' que es un puntero a float (flotante) retornando un puntero a float (flotante) y sólo es visible dentro del archivo donde es declarada. Y se la define retornando al argumento 'a'.
21. Se declara la variable 'a' que es un puntero a float (flotante) que se encuentra definida en un archivo externo.
22. Se declara la variable 'X' que apunta a un array de dos posiciones de tipo char definida en un archivo externo.
23. Se declara la función 'Z' que recibe como primer y único argumento a 'Y' que es un array de cinco posiciones de punteros a int.
24. Se declara y define la variable 'Z' de tipo long (entero largo) sólo visible dentro del archivo donde es declarada.

Escriba las siguientes definiciones/declaraciones:

1. Definición de un puntero a la función MULT que tome dos enteros largos con signo y devuelva su producto.

```
#include <stdio.h>
signed long int MULT(signed long int a, signed long int b) {
   return a*b;
}
void main() {
   signed long int (*ptrMULT) (signed long int, signed long int);
   ptrMULT = MULT;
   printf("3*4 = %ld\n", ptrMULT(3,4));
}
```

2. Declaración de un puntero a puntero a punto flotante de doble presición.
   - `extern double** a;`
3. Definición de un caracter sin signo sólamente visible en el módulo.
   - `static unsigned char;`
4. La declaración de una función denominada suma que tome como parámetros 2 punteros a entero y devuelva un puntero a un número de punto flotante de doble precisión.
   - `extern double* suma(int* a, int* b);`
5. La definición de un puntero a una función que toma como parámetros un entero corto con signo y un puntero a puntero a caracter y devuelve un número de punto flotante. El mismo debe poder accederse desde cualquier módulo del programa.

```
float (*ptrFuncion) (signed short int, char**);
ptrFuncion = funcion;
```

6. La definición de un entero con signo denominado A.
   - `signed int A;`
7. La definición de una función de alcance local al archivo de definición, denominada suma que tome como parámetros 2 enteros y devuelva un puntero a caracter con el resultado de la suma formateada como una cadena.

```
#include <stdio.h>
static char* suma(int a, int b) {
   int suma = a+b;
   static char cad[16];
   snprintf(cad, 16, "%d", suma);
   return cad;
}
void main() {
   char* str = suma(1,2);
   printf("%s\n", str);
}
```

8. La declaración de un puntero a una función que toma como parámetros un entero corto sin signo y un puntero a puntero a entero y devuelve un número de punto flotante.

```
extern float (*ptrFuncion) (signed short int, int**);
ptrFuncion = funcion;
```

9. La definición de un puntero a puntero a entero.

```
#include <stdio.h>
void main() {
   int c = 3;
   int* b = &c;
   int** a;
   a = &b;
   printf("%d\n", **a);
}
```

10. La declaración de una función de alcance local que toma como parámetros un puntero a caracter sin signo y un puntero a puntero a entero con signo; y no devuelve nada.
    - `static void funcion(unsigned char* a, signed int** b); //La declaramos y definimos ya que no se puede usar extern y static en simultaneo.`
11. La definición de un puntero a puntero a número de punto flotante de doble precisión, de alcance en el archivo donde se define.

```
#include <stdio.h>
void main() {
   static double c = 3.14;
   static double* b = &c;
   static double** a;
   a = &b;
   printf("%lf\n", **a);
}
```

12. La declaración de un puntero a una función que toma como parámetro un entero y un puntero a caracter sin signo; y devuelve un entero.

```
extern int (*ptrFuncion) (int, unsigned char*);
ptrFuncion = funcion;
```

13. Declaración de un puntero a función que tome 2 enteros cortos sin signo y devuelva un puntero a un arreglo de 10 caracteres.

```
extern char** (*ptrFuncion) (unsigned short int, unsigned short int);
ptrFuncion = funcion;
```

14. Definición de una variable local de una función, de tipo arreglo de punteros a nùmeros de puntos flotantes de doble precisión que conserve el valor entre llamados a la función.
    - `static double* a[arraySize];`
15. Definición de la Función suma, que tome un arreglo de 10 enteros y devuelva un puntero a entero con la suma.

```
#include <stdio.h>
int* suma(int a[10]) {
   static int sumatoria = 0; //Static para que persista el valor de s
   for (int i=0; i<10; i++)
      sumatoria += a[i];
   return &sumatoria;
}
void main() {
   int array[10] = {1, 2, 3, 4, 5 ,6 ,7, 8 ,9, 10};
   int* ptrSumatoria = suma(array);
   printf("Sumatoria de elementos de a[10] = %d\n", *ptrSumatoria);
}
```

16. Una definición de una variable entero largo sin signo, global, no visible fuera del fuente, llamada X.
    - `static unsigned long int X;`

17. Una definición de una función llamada LaFuncion que tome un arreglo de caracteres y devuelva un puntero a un arreglo de enteros cortos sin signo.

```
#include <stdio.h>
unsigned short int* LaFuncion(char* a) {
   static unsigned short int aUSInts[16];
   for (int i=0; i<16; i++) {
      if (a[i] == '\0')
         break;
      else
         aUSInts[i] = a[i] - 48; //ASCII conversion
   }
   return aUSInts;
}
void main() {
   unsigned short int* aUSInts = LaFuncion("1234");
   for (int i=0; i<4; i++)
      printf("%d\n", aUSInts[i]);
}
```

18. Una definición de un puntero a un arreglo de 10 entero sin signo, llamado ptrA.

```
unsigned int a[10] = {1,2,3,4,5,6,7,8,9,10};
unsigned int (*ptrA)[10];
ptrA = &a;
```

19. La declaración de una función de alcance local que toma como parámetros un entero y un puntero a puntero a caracter sin signo; y no devuelve nada.
    - `static void funcion(int a, unsigned char** b); //Declaración y definición.`
20. La declaración de un puntero a número de punto flotante de doble precisión.
    - `extern double* a;`
21. La definición de un puntero a una función que toma como parámetros un entero y un puntero a puntero a caracter sin signo; y no devuelve nada. El mismo debe poder accederse desde cualquier módulo del programa.

```
void (*ptrFuncion) (int, unsigned char**);  //En el header del archivo para que sea de acceso global.
ptrFuncion = funcion;
```

22. La declaración de una función que toma como parámetros un caracter y un puntero a puntero a caracter.
    - `void funcion(char a, char** b);`
23. La definición de un puntero a una función que toma como parámetros un caracter y un puntero a puntero a caracter. El mismo debe poder accederse desde cualquier módulo del programa.
    - `void (*ptrFuncion) (char, char**) = funcion;`
24. La declaración de un puntero a puntero a caracter.
    - `char** a;`
25. Declaración de un puntero a una función denominada C, que reciba como parámetros un entero largo sin signo y un entero corto con signo. Y que devuelva un puntero a caracter.

```
char* (*ptrC) (long unsigned int, short signed int);
ptrC = C;
```

26. Definición de un puntero a un número con punto flotante de doble precisión, denominado a.

```
#include <stdio.h>
int main() {
   double b = 3.14;
   double* a;
   a = &b;
   printf("%lf\n", *a);
}
```

27. Definición de un puntero a caracter sin signo, llamado ptrC.

```
#include <stdio.h>
int main() {
   unsigned char c = 'a';
   unsigned char* ptrC;
   ptrC = &c;
   printf("%c\n", *ptrC);
}
```

28. La declaración de una función que toma como parámetros un entero y un puntero a carácter.
    - `void funcion(int a, char* b);`
29. La declaración de un carácter llamado C.
    - `char C;`
30. La definición de un puntero local a una función que recibe un puntero a carácter y un número de punto flotante de doble precisión y devuelve un entero.

```
int (*ptrFuncion) (char*, double);
ptrFuncion = funcion;
```

31. La declaración de una función denominada suma que tome como parámetros 2 punteros a entero y devuelva un puntero a un número de punto flotante de doble precisión.
    - `double* suma(int* a, int* b);`
32. La definición de un puntero a una función que toma como parámetros un entero corto con signo y un puntero a puntero a caracter y devuelve un número de punto flotante. El mismo debe poder accederse desde cualquier módulo del programa.
    - `float (*ptrFuncion) (unsigned short int, char**);  //En el header del archivo para que sea global.`
    - `ptrFuncion = funcion;`
33. La definición de un entero con signo denominado A.
    - `signed int A;`
34. Una definición de una variable de punto flotante de doble precisión, global, no visible fuera del fuente, llamada X.
    - `static double X; //En el header del archivo para que sea global.`
35. La definición de la función Suma que tome 2 enteros y devuelva la suma de ambos. La función sólo debe ser visible en el archivo donde se la define.
    - `static int Suma(int a, int b) {return a+b;}`
36. Una declaración de la funcion main.
    - `int main(int argc, char** argv);`
37. La definición de una función de alcance local al archivo de definición, denominada suma que tome como parámetros 2 enteros y devuelva un puntero a caracter con el resultado de la suma formateada como una cadena.

```
static char* suma(int a, int b) {
        char *ptr = (char*) malloc(50 * sizeof(char)); int s = a+b;
        sprintf(ptr, "%d", s); return ptr;
}
```

Dentro del siguiente código:

```
int main(int argc, char *argv[]) {
        return 0;
}
```

Defina:

a) Un puntero a nro flotante de doble precisión, a alocarse en el stack.
b) Un Arreglo para albergar 4 caracteres que se aloque en el datasegment.
c) Un entero con signo dentro de la función main, a alocarse en el datasegment.

```
int main(int argc, char *argv[]) {
        // a)
        double* ptrDouble;
        // b)
        static char c[4];
        // c)
        static signed int;

        return 0;
}
```

Explique las características de las variables A1, A2, A3, A4, A5; indicando i) tipo, ii) area de memoria donde residen, iii) si deben o no liberarse, iv) tamaño que ocupan:

```
extern float A2;
MiFuncion(){
        static double A3;
        char A4;
        float *A5=new float;
        ....
}
static unsigned A1;
…
```

VARIABLE	TIPO	AREA EN MEMORIA	DEBE LIBERARSE?	TAMAÑO
A1	unsigned int	datasegment	no	4/8 bytes (en arquitectura de 32/64 bits)
A2	float	no asigna memoria, es solo un nombre		
A3	double	datasegment	no	8/16 bytes (en aequitectura de 32/64 bits)
A4	char	stack	no	1 byte
A5	float*	A5 en el stack y apunta al heap	si (memoria en heap)	1 byte A5 y 4/8 bytes (en arquitectura de 32/64 bits) a donde apunta

<simil>

Explique las características de las variables A,B,C,D y E; indicando i) tipo, ii) area de memoria donde residen, iii) si deben o no liberarse, iv) tamaño que ocupan:

```
static int A;
extern int B;
main () {
        static int C;
        int D;
        int *E=new int;
        ....
}
```

VARIABLE	TIPO	AREA EN MEMORIA	DEBE LIBERARSE?	TAMAÑO
A	signed int	data segment	no	4/8 bytes (en arquitectura de 32/64 bits)
B	signed int	depende de la declaracion real	depende	4/8 bytes (en arquitectura de 32/64 bits)
C	signed int	data segment	no	4/8 bytes (en arquitectura de 32/64 bits)
D	signed int	stack	no	4/8 bytes (en arquitectura de 32/64 bits)
E	signed int*	E en el stack y apunta al heap	si (memoria en heap)	1 byte E y 4/8 bytes (en arquitectura de 32/64 bits) a donde apunta

Escriba el .H correspondiente a una biblioteca que exporta:

- Una función 'DivEnt' que toma un puntero a un arreglo de 2 nros de punto flotante y retorna un puntero a entero.
- Una función llamada 'Valid' que no toma parámetros y devuelve un puntero a entero.
- Una variable de tipo entero con signo llamada ErrorCode.

```
#ifndef _ARCHIVO_H_
#define _ARCHIVO_H_
int* DivEnt(float* a[2]);
int* Valid();
signed int ErrorCode;
#endif /*_ARCHIVO_H_*/
```

<simil>

Escriba el .H correspondiente a una biblioteca que exporta:

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

```
#ifndef _ARCHIVO_H_
#define _ARCHIVO_H_
typedef struct alumno_t {
    char nombre[50];
    unsigned int padron;
} alumno_t;
unsigned int alumno_get_padron(alumno_t* a);
void procesar_alumnos(alumno_t* a, unsigned int b, void (*ptrF) (alumno_t*));
#endif /*_ARCHIVO_H_*/
```

Encapsulamiento, Constructores y Sobrecarga de operadores (19) (10/19)

Des-habilitar/Deletear constructores y operadores de copia

```
1.  //Constructor por copia des-habilitado
2.  ClaseA(const ClaseA&) = delete;
3.
4.  //Constructor por movimiento des-habilitado
5.  ClaseA(ClaseA&&) = delete;
6.
7.  //Operador de asignación des-habilitado
8.  ClaseA& operator=(const ClaseA&) = delete;
9.
10. //Operador de asignación por movimiento des-habilitado
11. ClaseA& operator=(ClaseA&&) = delete;
```

Prototipos de constructores y sobrecarga de operadores

```
1.  class Complejo {
2.  private:
3.      float real;
4.      float imaginario;
5.  public:
6.      //Constructores y operadores de asignacion
7.      Complejo(const float real=0, const float imaginario=0); //Constructor por defecto de Clase
8.      Complejo(const Complejo& copiable);                     //Constructor por Copia
9.      Complejo(Complejo&& copiable);                          //Constructor por movimiento
10.     Complejo& operator=(const Complejo& copiable);          //Sobrecarga del Operador asignación =
11.     Complejo& operator=(Complejo&& copiable);               //Sobrecarga del Operador asignación = por movimiento
12.
13.     //Sobrecarga de Operadores
14.     Complejo operator+(const Complejo& c2) const;           //Idem + - * /
15.     bool operator==(const Complejo& c2) const;              //Idem == != < > <= >=
16.     Complejo& operator++();                                 //prefix ++, ej ++a     Idem ++ --
17.     Complejo operator++(int);                               //postfix ++, ej a++    Idem ++ --
18.     operator bool() const;                                  //Operador de casteo, idem bool int float char double etc..
19.
20.     //Prototipos de sobrecarga de operadores globales
21.     //Opcion 1: indicar con friend la declaracion para darle acceso privado a la clase Complejo y evitar getters y setters.
22.     friend std::ostream& operator<<(std::ostream& output, const Complejo& c);
23.     friend std::istream& operator>>(std::istream& input, Complejo& c);
24. };
25.
26. //Prototipos de sobrecarga de operadores globales
27. //Opcion 2: no declaramos en clase Complejo y usamos getters y setters de Complejo para acceder a real e imaginario.
28. //cout es de la clase ostream y no podemos modificarla -> sobrecargamos el operador global
29. //cin es de la clase istream y no podemos modificarla -> sobrecargamos el operador global
30. //std::ostream& operator<<(std::ostream &output, const Complejo &c);
31. //std::istream& operator>>(std::istream &input, Complejo &c);
32.
33. //Tanto para opcion 1 u opción 2 se define de esta manera.
34. std::ostream& operator<<(std::ostream &output, const Complejo &c) {/*..*/}
35. std::istream& operator>>(std::istream &input, Complejo &c) {/*..*/}
```

Constructor por copia default y Operador Asignación default

En C y C++ pasa todos los objetos por copia por default.
En C++ si un objeto no tiene implementado un constructor por copia se le asigna un constructor por copia default (excepto que se deletee explicitamente el constructor por copia, en ese caso el objeto dejaría de ser copiable).
En C++ si un objeto no tiene implementada la sobrecarga del operador asignacion =, se le creará una implementacion default que realiza una copia bit a bit naive (excepto que se deletee explicitamente el constructor por copia, en ese caso el objeto dejaría de ser copiable).

La copia por default es una copia bit a bit naive que funciona bien para objetos simples, pero para objetos complejos puede ser necesaria una deep copy (copia profunda) que debe ser implementada.
Un ejemplo de objeto complejo sería que un objeto A apunte a un objeto B. Si se realiza una copia default (llamemosla C), ésta va también a apuntar a B. Lo cual podría ser o no lo requerido/deseado según la situación, aúnque en general, en la mayoria de los casos terminaría dando problemas.
Además de las relaciones A->B hay que ver con detenimiento las referencias a File Descriptors.

Si no queremos que un objeto sea copiable podemos:
1. Declarar de forma privada el constructor por copia y el operador asignacion (sin definirlos). El error de tratar de copiar se detecta en etapa de compilacion.
2. Deletear el constructor por copia y el operador asignacion. El error de tratar de copiar se detecta en etapa de compilacion.
3. Declarar de forma pública el constructor por copia y el operador asignacion y que en su definición se lanze una excepción. El error de tratar de copiar se detecta runtime.

Hasta antes de C++11 cuando se retornaba un objeto en una funcion se utilizaba el constructor por copia, pero que pasa si el objeto a retornar es muy costoso de construir?
Lo que se trata de hacer en C++11 con el Constructor por movimiento esque antes de que se destruya el objeto a retornar, se pasen todas las referencias de los datos y se le reasignen al nuevo objeto pero sin realmente crear todo el objeto de cero internamente (que es lo costoso).
El estándar garantiza que, de haber un constructor de movimiento disponible, será éste el invocado al construir el objeto retornado por la función, prefiriéndose dicha operación a la copia tradicional. Pasa lo mismo

Constructor de Clase | Constructor por Copia | Operador Asignación = | Constructor por Movimiento | Operador Asignacion = por Movimiento

```cpp
#include <iostream>

class A {
private:
    std::string nombre;
    std::string texto;
public:
    A(const std::string nombre);      //Constructor por defecto de Clase
    A(const A& copiable);             //Constructor por Copia
    A(A&& copiable);                  //Constructor por movimiento
    A& operator=(const A& copiable);  //Sobrecarga del Operador asignación =
    A& operator=(A&& copiable);       //Sobrecarga del Operador asignación = por movimiento
    void mostrar() const;
};

//Constructor por defecto de Clase
A::A(const std::string nombre) {
    this->nombre = nombre;
    this->texto = "me creé con el Constructor de Clase";
}

//Constructor por Copia
A::A(const A& copiable) {
    this->nombre = copiable.nombre;
    this->texto = "me creé con el Constructor por Copia";
}

//Constructor por movimiento
A::A(A&& copiable) {
    //Copio la informacion del objeto a destruir al nuevo objeto
    this->nombre = copiable.nombre;
    this->texto = "me creé con el Constructor por Movimiento";
    //Libero los recursos del objeto que se va a destruir
    copiable.nombre = "";
    copiable.texto = "";
}

//Sobrecarga del Operador asignación =
A& A::operator=(const A& copiable) {
    this->nombre = copiable.nombre;
    this->texto = "me reasigné con el Operador Asignación =";
    return *this;
}

//Sobrecarga del Operador asignación = por movimiento
A& A::operator=(A&& copiable) {
    //Copio la informacion del objeto a destruir al nuevo objeto
    this->nombre = copiable.nombre;
    this->texto = "me reasigné con el Operador Asignación = por movimiento";
    //Libero los recursos del objeto que se va a destruir
    copiable.nombre = "";
    copiable.texto = "";
    return *this;
}

void A::mostrar() const {
    std::cout << "Mi nombre es " << this->nombre << " y " << this->texto <<std::endl;
}

A foo(std::string nombre) {
    A a(nombre);
    return std::move(a); //Invoco al constructor por movimiento
}

int main() {
    A a1("Juan");       //Aplica el Constructor de Clase
    a1.mostrar();       // >>> "Mi nombre es Juan y me creé con el Constructor de Clase"

    A a2 = a1;          //Aplica el Constructor por Copia
    a2.mostrar();       // >>> "Mi nombre es Juan y me creé con el Constructor por Copia"

    A a3("Tomas");      //Aplica el Constructor de Clase
    a3.mostrar();       // >>> "Mi nombre es Tomas y me creé con el Constructor de Clase"
    a3 = a1;            //Aplica el Operador Asignación =
    a3.mostrar();       // >>> "Mi nombre es Juan y me reasigné con el Operador Asignación ="

    A a4 = foo("Teo");  //Aplica el Constructor por Movimiento
    a4.mostrar();       // >>> "Mi nombre es Teo y me creé con el Constructor por Movimiento"

    A a5("Jeronimo");   //Aplica el Constructor de Clase
    a5.mostrar();       // >>> "Mi nombre es Jeronimo y me creé con el Constructor de Clase"
    a5 = foo("Teo");    //Aplica el Operador Asignación = por Movimiento
    a5.mostrar();       // >>> "Mi nombre es Teo y me reasigné con el Operador Asignación = por Movimiento"
    return 0;
}
```

Notas del código:

- Los constructores nunca son métodos const. (Crean el objeto, no tendría sentido).
- El parámetro &&copiable del constructor por movimientoy el operador por movimiento no debe ser const ya que se debe modificar al mismo para limpiar su memoria dentro del método.

Describa el prototipo de los siguientes operadores para que conserven la semántica esperada:

- operator>
- operator=
- operator int
- operator+
- operator++

Ver Resumen con los prototipos de sobrecarga de todos los operadores

¿En qué casos recomienda Ud. incluir un **Constructor de Copia** en una clase? ¿Qué sucedería si no lo incluye? **Justifique mediante un ejemplo.**

Recomendaría utilizar un constructor por copia cuando se requiere hacer una deep copy, es decir una copia profunda que el compilador no sabe resolver. Por ejemplo si tenemos un objeto que uno de sus atributos es un puntero a otro objeto, podríamos querer que cuando se realice una copia del objeto se duplique también el objeto al que se apunta y no que ambas copias apunten al mismo objeto.

¿Por qué es importante utilizar el modificador const en, por ejemplo, los operadores <, >, <=, bool?

Para el caso de los operadores de comparación (>,<,<=,>=,==,!=) se debe asegurar que ambos miembros de la comparación no sean modificados.
Prototipo del operador >: const bool operator>(const Complejo& c2) const;
bool r = a<b; // 'r' es el resultado de la comparación, 'a' es el 1er miembro, 'b' el 2do miembro
El 1er const indica que el resultado es una constante (no es 100% necesario).
El 2do const indica que el 2do miembro de la comparación no será modificado (esencial).
El 3er const indica que el 1er miembro de la comparación no será modificado (esencial).

En el caso de los operadores de casteo (ej: bool) querémos asegurar que el objeto casteado no sea alterado al representarse como un bool.
Prototipo del operador bool: operator bool() const;

(3)

Para utilizar una clase X con las librerías STD se piden algunas características particulares (Contructor default, operador ==, etc.) ¿por qué cree que estas son requeridas?¿que sucedería si no se proveen las mismas? Justifique.
¿Qué es necesario que **tenga una clase** para que pueda ser utilizada en **std::list**? **Justifique.**
¿Qué es necesario que **tenga una clase** para que pueda ser utilizada en **std::queue**? **Justifique**.

Se requieren ciertas características (como la sobrecarga de ciertos operadores), para poder utilizar las funciones que provee la librería sin inconvenientes.
Por ejemplo, si queremos utilizar nuestra clase X con algún contenedor STL como por ejemplo std::vector<X>, es necesario que nuestra clase implemente por ejemplo el operador ==, que es utilizado por varios métodos de la clase std::vector<X> que realizan comparaciones.
Si no se implementan las características necesarias nuestro código no va a compilar.

Hablando en términos genéricos ¿Qué **recomendación** especial tendría Ud. al momento de escribir un **operador=**? Ejemplifique.
¿Qué cuidado especial tendría Ud. al momento de escribir un **operador=** para una **clase que usa memoria dinámica?** Ejemplifique.

Hay que tener cuidado si la clase que copiamos usa memoria dinámica. Imaginemos por ejemplo que en el constructor de la ClaseA se reserva memoria para un atributo.
Clase A a;
Clase B b;
a = b; // Ahora el atributo de a apunta al atributo de b, con lo cual si hacemos delete a.

Respuesta de internet:
El operator = es como los demás. C++ por defecto tiene el operador igual definido para clases del mismo tipo. Por ejemplo, sin necesidad de redefinir nada, podemos hacer:

```
ComplejoC a;
ComplejoC b;
a = b;
```

El operador = por defecto copia el contenido del uno sobre el otro (byte a byte), útil en clases sencillas, pero insuficiente en clases más complejas.
Si algún atributo es un puntero, tenemos que tener cuidado con lo que hacemos. Supongamos que ClaseC tiene un atributo Atributo que es un puntero. Supongamos también que en el constructor de la clase, se hace new del puntero para que tenga algo y en el destructor se hace el delete correspondiente.

```
ClaseC {
  public:
    ClaseC () {
      Atributo = new int[3]; // Se crea un array de tres enteros
    }
    ~ClaseC () {
      delete [] Atributo; // Se libera el array.
    }
  protected:
    int *Atributo;
};
```

Si ahora hacemos esto:

```
ClaseC *a = new ClaseC(); // a tiene ahora un array de 3 enteros en su interior
ClaseC *b = new ClaseC(); // b tiena ahora otro array de 3 enteros en su interior
*a = *b; // Se copia el Atributo de b sobre el de a, es decir, ahora a->Atributo apunta al mismo sitio que b->Atributo
delete b; // Ahora si que la hemos liado.
```

Cuando hacemos a=b, con el operador igual por defecto de C++, se hace que el puntero Atributo de a apunte al mismo sitio que el de b. El array original de a->Atributo lo hemos perdido, sigue ocupando memoria y no tenemos ningún puntero a él para liberarlo.
Cuando hacemos delete b, el destructor de b se encarga de liberar su array. Sin embargo, al puntero a->Atributo nadie le avisa de esta liberación, se queda apuntando a una zona de memoria que ya no es válida. Cuando intentemos usar a->Atributo, puede pasar cualquier cosa (cambios aleatorios de variables, caidas del programa, etc).

La forma de solucionar esto, es definiendo nosotros un operator = que haga una copia real del array, liberando previamente el nuestro o copiando encima los datos.

```
ClaseC {
  public:
    ClaseC &operator = (const ClaseC &original) {
      int i;
      /* Damos por supuesto que ambos arrays existen y son de tamaño 3 */
      for (i=0;i<3;i++)
        Atributo[i] = original.Atributo[i];
    }
};
```

1) **(2)**
Declare la clase COMPLEJO, incluyendo operadores aritméticos, lógicos y de entrada/salida.
<simil>
Declare una clase Complejo para encapsular un nro. complejo. Incluya al menos: Constructor default, Constructor con valores de inicialización y Constructor de Copia; Operador <, ==, =, int (que devuelva la parte real del nro. encapsulado) y << (impresión). Implemente el operator<<.
Solución aquí, solución (con definiciones) en 1.cpp

```cpp
#include <iostream>
class Complejo {
private:
    float real;
    float imaginario;
public:
    //Constructores y operadores de asignacion
    Complejo(const float real = 0, const float imaginario = 0); //Constructor default de Clase (y con valores de inicializacion)
    //Complejo(const Complejo& copiable); //Constructor de copia
    //Complejo& operator=(const Complejo& copiable); //Sobrecarga de Operador Asignacion=
    //Sobrecarga de Operadores
    Complejo operator+(const Complejo& c2) const;
    Complejo operator-(const Complejo& c2) const;
    bool operator==(const Complejo& c2) const;
    bool operator!=(const Complejo& c2) const;
    bool operator<(const Complejo& c2) const;
    Complejo& operator++();                                 //prefix ++, ej ++a
    Complejo operator++(int);                               //postfix ++, ej a++
    operator bool() const;
    operator int() const;
    //Operadores globales con acceso privado a ésta.
    friend std::ostream& operator<<(std::ostream& output, const Complejo& c);
    friend std::istream& operator>>(std::istream& input, Complejo& c);
};

//Sobrecarga de operadores globales << y >> con acceso privado a la clase Complejo
//friend va sólo en la declaracion en la clase Complejo
std::ostream& operator<<(std::ostream& output, const Complejo& c) {
    char signo = '+';
    float r = c.real;
    float i = c.imaginario;
    if (i<0) { //Signo - e invierno i para que quede positivo
        signo = '-';
        i = -i;
    }
    output << std::to_string(r) << signo << std::to_string(i) << "i" ;
    return output;
}

//friend va sólo en la declaracion en la clase Complejo
std::istream& operator>>(std::istream& input, Complejo& c) {
    input >> c.real >> c.imaginario;
    return input;
}
```

2) Declare la clase Número para almacenar un número de 100 cifras decimales. Incluya: constructor default, constructor de copia, los operadores +, ++ (posfijo), ++ (prefijo), >, =, << (corrimiento de bits) y <<(impresión). Implemente el operador <<(impresión).

```cpp
#include <iostream>
class Numero {
public:
    //Constructores: nunca retornan nada ni son metodos constantes
    Numero(double valor=0); //Constructor default
    Numero(const double& n); //Constructor por copia
    //Comparadores: const n para preservar el objeto original, retorna Numero& porque se modifica el 1er miembro.
    Numero& operator=(const Numero& n);  //Sobrecarga operador asignacion=
    bool operator>(const double& n) const;  //Sobrecarga operador >
    //Binarios aritmeticos, , ambos const es para preservar ambos miembros de la suma. Se retorna Numero porque se genera uno nuevo.
    Numero operator+(const double& n) const; //Sobrecarga operador +
    //Operadores de Incremento y decremento
    Numero& operator++(); //prefix, incrementamos *this y luego lo retornamos
    Numero operator++(int); //postfix, copiamos *this, incrementamos *this y luego retornamos la copia
    std::string getValor() const;
};

/*No se puede editar la clase std::cout ni std::cin entonces editamos
 * el operador global << y >> y pasamos de paràmetro ambos miembros.
 */
std::ostream& operator<<(std::ostream& os, const Numero& n); //n es const porque queremos preservar el dato.
std::istream& operator>>(std::istream& is, Numero& n); //n no es const porque se va a cargar informacion en él.

std::ostream& operator<<(std::ostream& os, const Numero& n) {
    os << n.getValor();
    return os;
}
```

Nota: resolví usando opcion 2 para los operadores << e >>

3) Declare la clase Email para encapsular una cadena correspondiente a una dirección de email. Incluya al menos: Constructor default y Constructor de Copia; Operador <<, ==, =, int y >>. Implemente el operador >>.

```cpp
#include <iostream>
class Email {
private:
    std::string email;
public:
    Email(const std::string = "");
    Email(const Email& e);
    Email& operator=(const Email& e);
    bool operator==(const Email& e) const;
    operator int() const;
    friend std::ostream& operator<<(std::ostream& os, const Email& e);
    friend std::istream& operator>>(std::istream& is, Email& e);
};

//Suponemos que es para operaciones de entrada salida
std::istream& operator>>(std::istream& is, Email& e) {
    is >> e.email;
    return is;
}
```

Nota: resolví usando opcion 1 para los operadores << e >>

4) Declare una clase LEGAJO para encapsular un nro de legajo de un empleado. Incluya al menos: Constructor default, Constructor con string de inicialización y Constructor de Copia; Operador <, ==, = y << (impresión). Implemente el operator>.

```cpp
#include <iostream>
class Legajo {
private:
    int legajo;
public:
    //Constructores
    Legajo(const std::string legajo);       //Constructor default de clase
    Legajo(const Legajo& l);                //Constructor por copia
    //Sobrecarga operador asignacion
    Legajo& operator=(const Legajo& l);     //Operador asignacion=
    //Sobrecarga operadores de comparacion
    bool operator<(const Legajo& l) const;
    bool operator==(const Legajo& l) const;
    //Sobrecarga operadores << >> de cout y cin
    friend std::ostream& operator<<(std::ostream& os, const Legajo& l);
    friend std::istream& operator>>(std::istream& is, Legajo& l);
};

bool Legajo::operator<(const Legajo& l) const {
    if (this->legajo < l.legajo)
        return true;
    return false;
}
```

Nota: resolví usando opcion 1 para los operadores << e >>

5) Declare una clase a elección que incluya los operatores +, >, >> y bool.

```cpp
#include <iostream>
class A {
public:
    A();
    A operator+(const A& a) const;
    bool operator>(const A& a) const;
    operator bool() const;
    friend std::istream& operator>>(stD::istream& is, A& a); //Hipotesis: suponemos que la necesitan para entrada cin
};
```

6) La clase “ORACION” utiliza un puntero (char *a) para almacenar un string terminado en null. Escriba la declaración de esta clase no olvidando: Constructor default, Constructor de copia, operadores +,-, >>, <<. Implemente el operador - de forma que elimine de la primer cadena todas las ocurrencias de la segunda.

7) Declare una clase Patente para encapsular una patente de vehículo de Argentina. Incluya al menos: Constructor default, Constructor con valor de inicialización y Constructor de Copia; Operador <, ==, =, int (que devuelva la parte entera de la patente) y << (impresión). Implemente el operator>.

Nota: en 7.cpp están implementados más metodos.

```cpp
#include <iostream>
class Patente {
private:
    std::string patente;
public:
    Patente(const std::string p="AAA000");
    Patente(const Patente& p);

    bool operator<(const Patente& p) const;
    bool operator>(const Patente& p) const;
    bool operator==(const Patente& p) const;
    operator int();
    friend std::ostream& operator<<(std::ostream& os, const Patente& p);
};

bool Patente::operator>(const Patente& p) const {
    if (this->patente>p.patente)
        return true;
    return false;
}
```

Declare la clase CADENA para encapsular un string. Incluya, mínimamente, el constructor default, el constructor de copia, los operadores >,==,+, int y >>. Defina el operador >>.

Declare una clase BitString para encapsular una secuencia de 200 bits. Incluya al menos: Constructor default, Constructor con valores de inicialización y Constructor de Copia; Operador <, ==, =, int y >> (carga desde STDIN). Implemente el operador>>.

Declare una clase BigInt para encapsular un entero de 128Bits. Incluya al menos: Constructor default, Constructor con valor de inicialización y Constructor de Copia; Operador <, ==, =, int y << (impresión). Implemente el operator+.

La clase SerieMatemática encapsula una serie de números de punto flotante. Implemente: Operador + (suma miembro a miembro los valores de una lista, asumiendo 0 para valores faltantes), Operador==,Operador int (devuelve el promedio de la serie) .

```
class SerieMatemátiva {
private:
int cant;
float *Lista;
public:
SerieMatemátiva() {cant=0;Lista=NULL;}
...
}
```

Declare la clase FechaYHora para encapsular una fecha y hora. Incluya al menos:

- Constructor default (fecha de hoy y hora actual),
- Constructor con string de inicialización,
- Constructor de Copia,
- Operador <,
- Operador ==,
- Operador =,
- Operador int (segundos desde 01/01/1970)

Declare e implemente los operadores << y >> para que la clase FechaYHora de este examen pueda imprimirse y cargarse en la consola.

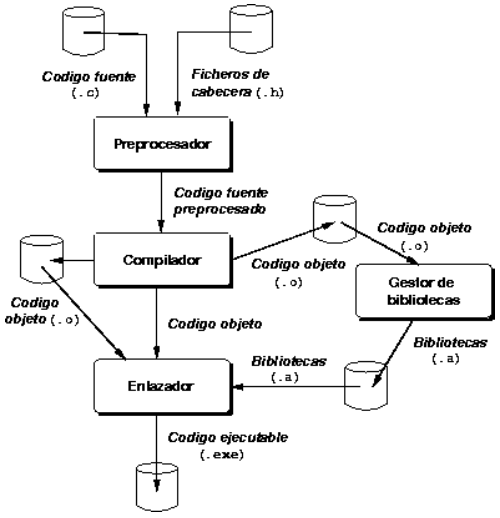
Lenguaje C/C++ (Teoría)

Compilacion / Precompilacion (19) (8/9)

(2) Describa el proceso de transformación de código fuente a un ejecutable. Precise las etapas y las tareas desarrolladas 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:
   1. Eliminar los comentarios.
   2. 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 traduciendolos y generando un fichero que contiene el código objeto (ficheros .o).
   Sub etapas de la compilación:
   1. Parseado del código fuente par armar un árbol semántico.
   2. Traducción de los simbolos parseados en instrucciones de ensamblador.
   3. 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.



(3) ¿En qué consiste el proceso de **precompilación**?

El proceso de precompilación o preprocesamiento es el encargado de dos tareas:

1. Sustitución y control de la compilación.
   La capacidad de incluir o no bloques de código dentro de nuestro archivo fuente. Las directivas más importantes : define, include, undef, if, else, endif, ifndef, ifdef...
2. Expansión de macros.

(3) Describa y ejemplifique el uso de las siguientes instrucciones de precompilación:

- #define
- #undef
- #ifdef
- #ifndef
- #include

La directiva #if, con las directivas #elif, #else y #endif, controla la compilación de partes de un archivo de código fuente. Si la expresión que se escribe (después de #if) tiene un valor distinto de cero, el grupo de líneas inmediatamente después de la directiva #if se conserva en la unidad de traducción. Lo mismo con #ifdef e #ifndef.

#define (definir símbolo) #undef (anular definición de símbolo)
La directiva #define se utiliza para definir un símbolo. Si utiliza el símbolo como expresión que se pasa a la directiva #if, la expresión se evaluará como true. A su vez, cada vez que el precompilador encuentre un match a ese simbolo, reemplazara por el valor asociado a dicho simbolo (Macros). A un simbolo definido por define se le puede asociar tanto como sentencias de codigo como valores literales que el
precompilador hara un reemplazo literal del mismo.

La directiva #undef permite anular la definición de un símbolo, de tal modo que si se utiliza como expresión de una directiva #if, la expresión se evaluará como false.

```
#define DEBUG
#undef TRACE
using System;

public class TestDefine {
  static void Main() {
    #if (DEBUG)
      Console.WriteLine("Debugging is enabled.");
    #else
      Console.WriteLine("Debugging is not enabled.");
    #endif

    #if (TRACE)
      Console.WriteLine("Trace is enabled.");
    #else
      Console.WriteLine("Trace is not enabled.");
    #endif
  }
}
// Output:
// Debugging is enabled.
// Trace is not enabled.
```

#ifndef (If not defined) #ifdef (If defined)
La instruccion de precompilacion ifndef le indica al preprocesador que busque en su tabla de simbolos si existe el simbolo.
Si la condicion es verdadera:, entonces se pondra el codigo que este entre #ifndef y:
a) Si hay else, entonces se pondrá el código entre $ifndef y #else y se ignora lo que figure en #else y #endif
b) si no hay else entonces se pondrá el código entre #ifndef y #endif
Y si la condición es falsa:
a) Si hay else, entonces se pondrá el código entre #else y #endif y se ignora lo que figure en #ifndef y #else
b) si no hay else entonces se ignorará todo el código entre #ifndef y #endif

Se puede utilizar para no caer en declaraciones circulares en los headers.
Si no se utilizaran los if not defined caeriamos en declaraciones circulares.
La primera vez que se incluye "algo.h", el simbolo "_ALGO_H_" no estara definido, entonces se define el simbolo "_ALGO_H_" y se procede a incluir las declaraciones. Con el #endif se indica que termina el if.
Cuando se incluya "otro.h" pasará lo mismo que antes, como _OTRO_H_ aún no se encontraba definido entrara al if por lo que se volverá a querer incluir "algo.h"
Ahora, si se vuelve a incluir "algo.h", la pregunta #ifndef _ALGO_H_ sera false porque el simbolo ya se definio previamente, por lo que no se volveran a incluir las declaraciones del archivo de cabecera cortando las declaraciones circulares.

```
/* algo.h */
#IFNDEF _ALGO_H_
#DEFINE _ALGO_H_
#INCLUDE “otro.h”
   // sarasa
#ENDIF
```

```
/* otro.h */
#IFNDEF _OTRO_H_
#DEFINE _OTRO_H_
#INCLUDE “algo.h”
   // sarasa
#ENDIF
```

#if chequea el valor del símbolo (true o false), mientras que #ifdef / #ifndef chequea la existencia del símbolo (sin importar si valor).

```
#define FOO 0
#if FOO
  // won't compile this
#endif
#ifdef FOO
  // will compile this
#endif
```

(2) ¿Qué significado tienen los siguiente símbolo de **PRECOMPILACIÓN (MACRO)**?
__LINE__
__FILE__
De un ejemplo **coherente** de su uso.

```
__LINE__   Número de la línea actual del código fuente (valor entero).
__FILE__   Nombre del archivo que se está compilando (cadena de caracteres).
__DATE__   Fecha de compilación del archivo fuente (cadena con formato "Mes DD AAAA").
__TIME__   Hora en que comenzó a compilarse (cadena de la forma "HH:MM:SS").
__STDC__   Se define como valor 1 si el compilador se ajusta al estándar ANSI.
```

__LINE__	__FILE__

`#include <stdio.h>`
`int main () {`
`  printf ("This is line %d.\n", __LINE__);`
`  return 0;`
`}`
`//Output: "This is line 3."`
	`#include <stdio.h>`
	`int main () {`
	`  printf ("This is line %d of file \"%s\".\n", __LINE__, __FILE__);`
	`  return 0;`
	`}`
	`//Output: "This is line 3 of file "\...\archivo.c"`

¿Qué tipos de errores son detectados en la etapa de **linkedición**? **Describa** cada uno de ellos especificando su causa.
<simil>
Describa con precisión que tareas lleva a cabo un linkeditor. Haga referencia a las definiciones y declaraciones hechas en el proyecto, y a las ambigüedades que pueden producirse. Ejemplifique mediante código sencillo.
Linker (montador o enlazador): Es el programa encargado de insertar al programa objeto el código máquina de las funciones de las librerías (archivos de biblioteca) usadas en el programa y realizar el proceso de montaje, que producirá un programa ejecutable .exe. Las librerias son una colección de código (funciones) ya programado y traducido a código máquina, listo para utilizar en un programa y que facilita la labor del programador.
En la etapa de linkedición se puede detectar funciones o variables que no hayan sido resueltos. Si estos simbolos no pueden ser hallados el linker lanza un error.
Hay dos formas de linkear, estatica o dinamicamente. El ejecutable estático es mucho mas pesado pero tiene todas las librerías incorporadas en el ejecutable, es decir que no tiene dependencias. En cambio, el linkeo dinámico genera un ejecutable más pequeño pero con dependencias que se resuelven en tiempo de ejecución.

(2)
¿Qué es una **ambigüedad** de compilación**? Ejemplifique.**
¿Qué **ambigüedades** pueden producirse en C++ al momento de decidir que **versión de un método se debe invocar? Ejemplifique**.
Ambigüedad de compilacion es cuando no se puede resolver una llamada a funcion, método o variable porque hay más de un identificador representando cosas distintas.
En el caso de C++ puede darse una ambiguedad de compilación al invocar un método de clase cuando por ejemplo se utiliza derivación múltiple y se heredan de dos clases base dos metodos que se llaman igual.

Ejemplo:

```
1.  #include <iostream>
2.  class ClaseA {
3.  public:
4.      std::string saludo() const { return "Saludo A"; }
5.  };
6.
7.  class ClaseB {
8.  public:
9.      std::string saludo() const { return "Saludo B"; }
10. };
11.
12. class ClaseC : public ClaseA, public ClaseB {};
13.
14. int main() {
15.    ClaseC CC;
16.    //std::cout << CC.Saludo() << std::endl; //Produce error de compilación por ambigüedad.
17.    std::cout << CC.ClaseA::saludo() << std::endl; //Resolvemos ambiguedad
18.    return 0;
19. }
```

(4) ¿Qué es la **Compilación Condicional**? ¿Qué uso tiene? ¿En qué parte del **proceso de transformación** de código se resuelve? **Ejemplifique** mediante código C/C++ dando un caso de utilidad.
Las directivas de compilación condicional permiten incluir o excluir de forma condicional partes de un archivo de código fuente.
La transformación de código se realiza en la etapa de preprocesado.
Excepto por las directivas de preprocesamiento, el código fuente omitido no se somete al análisis léxico.

Se utiliza por ejemplo para evitar caer en declaraciones circulares.

`/* algo.h */`
`#IFNDEF _ALGO_H_`
`#DEFINE _ALGO_H_`
`#INCLUDE “otro.h”`
`  // sarasa`
`#ENDIF`
	`/* otro.h */`
	`#IFNDEF _OTRO_H_`
	`#DEFINE _OTRO_H_`
	`#INCLUDE “algo.h”`
	`  // sarasa`
	`#ENDIF`

Otro ejemplo sería utilizar la directiva #DEFINE DEBUG para someter el código a DEBUG con esta directiva.

¿Cómo se **evita que un archivo .h sea incluido más de una vez** en el mismo módulo de compilación? **Ejemplifique**

Se puede evitar utilizando las siguientes instrucciones de precompilación #ifndef, #define y #endif
La primera vez que se incluye "algo.h", el simbolo "_ALGO_H_" no estara definido, entonces se define el simbolo "_ALGO_H_" y se procede a incluir las declaraciones. Con el #endif se indica que termina el if.
Cuando se vuelva a incluir "algo.h", la pregunta #ifndef _ALGO_H_ sera false porque el simbolo ya se definió previamente, por lo que no se volveran a incluir las declaraciones del archivo de cabecera cortando las declaraciones circulares.

```
/* algo.h */
#IFNDEF _ALGO_H_
#DEFINE _ALGO_H_
#INCLUDE “otro.h”
   // sarasa
#ENDIF
```

Macros (7) (3/4)

(3) ¿Qué es una **MACRO**? ¿Qué usos tiene? ¿En qué parte del **proceso de transformación** de código se resuelve?

Una MACRO es básicamente un alias que podemos incluir en nuestro código el cual será reemplazado por lo que hayamos definido cuando el compilador efectúe el paso de preprocesamiento. Las macros son capaces de realizar decisiones lógicas, o funciones matemáticas.

```
#define WRONG(A) A*A*A          /* Macro incorrecto para el cubo */
#define CUBE(A) (A)*(A)*(A)     /* Macro correcto para el cubo */
#define SQUR(A) (A)*(A)         /* Macro correcto para el cuadrado */
#define START 1
#define STOP  9
```

¿Qué **recomendaciones** daría Ud. al momento de escribir una **MACRO con fórmulas matemáticas? Justifique mediante ejemplos.**

Se recomienda el uso de parentesis para aislar las expresiones mateáticas en una MACRO.
Ejemplo:

```
#define CUBE_WRONG(A) A*A*A     /* Macro incorrecto para el cubo */
#define CUBE(A) (A)*(A)*(A)     /* Macro correcto para el cubo */
```

```
MACRO               MACRO EXPANDIDA       RESULTADO            RESULTADO ESPERADO
CUBE_WRONG(5+1)     5+1*5+1*5+1           16                   216
CUBE(5+1)           (5+1)*(5+1)*(5+1)     216                  216
```

<simil>

¿Por qué se recomienda encerrar los parámetros de una MACRO de C entre paréntesis (ej.: #define SUMA(a,b) (a)+(b) ? Ejemplifique.

Se recomienda utilizar paréntesis ya que 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.
En el caso de una macro SUMA no afecta ya que la suma (junto con la resta) son las últimas operaciones arítméticas que se ejecutan.
Pero si definieramos una macro MULTIPLICA si podrían darse efectos indeseados si no se utilizan los parentesis.

```
#define MULTIPLICA(a,b) (a)*(b)
#define MULTIPLICA_MAL(a,b) a*b
```

```
MACRO                          MACRO EXPANDIDA       RESULTADO            RESULTADO ESPERADO
MULTIPLICA_MAL(1+2,1+2)        1+2*1+2               5                    9
MULTIPLICA(1+2,1+2)            (1+2)*(1+2)           9                    9
```

Qué sucede si se escribe una **macro recursiva**?¿cómo se expande?

Escriba una macro DONDE que imprima por salida standard el nombre del archivo fuente y la línea donde fue utilizada.

```
#include <stdio.h>
#define DONDE printf("Archivo %s - Linea %d \n", __FILE__, __LINE__);
int main(int argc, char** argv) {
          DONDE;
          return 0;
}
```

Caracteristicas generales del lenguaje (6) (1/4)

¿Qué es la POO (**Programación Orientada a Objetos**)? ¿Qué **ventajas** ofrece su uso?.

El paradigma orientado a objetos (OO) define los programas en términos de comunidades de objetos. Los objetos son instancias de clases que definen las características comunes entre los objetos de una misma clase. Los objetos son entidades que combinan un estado (es decir, datos) y un comportamiento (esto es, procedimientos o métodos). Estos objetos se comunican entre ellos para realizar tareas. Es en este modo de ver un programa donde este paradigma difiere del paradigma imperativo o estructurado, en los que los datos y los métodos están separados y sin relación. El paradigma OO surge para solventar los problemas que planteaban otros paradigmas, como el imperativo, con el objeto de elaborar programas y módulos más fáciles de escribir, mantener y reutilizar. Entre los lenguajes que soportan el paradigma OO están Smalltalk, C++, Delphi (Object Pascal), Java y C#.

Ventajas:

- Reusabilidad: Clases bien diseñadas se pueden utilizar en distintas partes del programa y en numerosos proyectos.
- Mantenibilidad: Son más sensillos de leer y comprender ya que permiten ocultad detalles de implementación.
- Modificabilidad: La facilidad de añadir, suprimir o modificar nuevos objetos nos permite hacer modificaciones de una forma muy sencilla.
- Fiabilidad: Al dividir el problema en partes es más facil resolverlo y detectar posibles errores.

(2) ¿Qué **características** posee la **programación orientada a eventos**? ¿Cómo se programa en dicho paradigma?
¿Que caracteriza a una aplicación **Orientada a Eventos?**

La programación dirigida por eventos es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.

Para entender la programación dirigida por eventos, podemos oponerla a lo que no es: mientras en la programación secuencial (o estructurada) es el programador el que define cuál va a ser el flujo del programa, en la programación dirigida por eventos será el propio usuario —o lo que sea que esté accionando el programa— el que dirija el flujo del programa. Aunque en la programación secuencial puede haber intervención de un agente externo al programa, estas intervenciones ocurrirán cuando el programador lo haya determinado, y no en cualquier momento como puede ser en el caso de la programación dirigida por eventos.

Enumere **3 diferencias** entre C y C++. Ejemplifique

Paradigma de programación:
- C es un lenguaje basado en el paradigma Imperativo/Estructurado.
- C++ es un lenguaje basado en el paradigma orientado a objetos (admite también programación estructurada).

Flujos de Entrada/Salida (IO Input/Output):
- En C se utilizan las funciones printf y scanf para los flujos IO.

```c
#include <stdio.h>
int main() {
    int numero;
    scanf("%d",&numero);
    printf("El numero es %d \n",numero);
    return 0;
}
```

- En C++ se utiliza std::cout y std::cin para los flujos IO.

```cpp
#include <iostream>
int main() {
    int numero;
    std::cin >> numero;
    std::cout << "El numero es " << numero << std::endl;
    return 0;
}
```

(2) ¿Que características posee el lenguaje **C** que le permiten llamarse **portable**?

Cuando hablamos de portabilidad de aplicaciones nos referimos a aplicaciones que pueden ejecutarse en varias plataformas de software de base, o, en definitiva, en varios sistemas operativos, como Unix, Linux o Windows.
Una aplicación escrita en C es portable a nivel de código fuente, es decir, podemos portar el fuente de la aplicación de un sistema a otro, y compilarla para obtener los ejecutables para dicho sistema, sin mayores modificaciones. Por supuesto, deberemos tener instaladas las bibliotecas de desarrollo utilizadas, en el sistema operativo en el que vamos a compilar.

Atributos estáticos (8) (5/5)

¿Qué significado tiene la palabra reservada **static** cuando es antepuesta a una variable global**. Ejemplifique su uso.**
Las variables globales se almacenan en el Data Segment. La vida de la variable es igual a la del programa.
Al ser estática no puede ser accedida desde otros achivos utilizando extern.

¿Qué significado tiene la palabra reservada **static** cuando es antepuesta a una variable local de una funcion**. Ejemplifique su uso.**
El valor de la variable var_estatica no se pierde aunque ya no se encuentre dentro del alcance, es decir, cuando el programa haya salido de la función en que la variable fue definida. Esto diferencia una variable estática de una variable no estática, cuyo valor se pierde al final de la función.

(2) Explique qué son **los métodos static** y para qué sirven. De un breve ejemplo donde su uso sea imprescindible.
Métodos de clases también pueden ser estáticas. Declarar un método estático es una promesa que este método sólo utiliza miembros estáticos de la clase. Este tipo de método no conoce el puntero this, por lo cual no se puede hacer directamente referencia a métodos y variables no estátitocs de la clase.
Ver https://trucosinformaticos.wordpress.com/2010/12/05/como-usar-static-en-c-y-c/

```
1.  class MiClase {
2.  public:
3.      static int una_variable_estatica; // Una variable estática
4.      int una_variable_no_estatica;     // Una variable no estática
5.
6.      // Un método estático
7.      static void haz_algo(void) {
8.          // Se pueden usar variables estáticas
9.          una_variable_estatica = 5;
10.         // Error: no se pueden usar variables que requieren un puntero this
11.         una_variable_no_estatica = 5;  // Error de compilación
12.     }
13.
14.     // Un método normal (no estático)
15.     void haz_otra_cosa(void) {
16.         // Se pueden usar variables y métodos estáticos también en métodos no estáticos
17.         una_variable_estatica = 5;
18.         haz_algo();
19.         // Una variable de instancia en un método de instancia
20.         una_variable_no_estatica = 5;  // Correcto
21.     }
22. };
```

Explique el significado de la palabra static cuando es antepuesta a una funcion de un archivo.
Las funciones estáticas son funciones que solo son visibles para otras funciones del mismo archivo.

(3)
¿Qué son las **variables de clase**?¿qué **características** tienen?¿cómo se **declaran**?¿cómo se **definen**? Ejemplifique mediante un sencillo **ejemplo**.
<simil>
Explique **un uso posible** de las **atributos/variables de clase. Ejemplifique**.
<simil>
¿Qué **son los atributos estáticos**? **Ejemplifique** su uso mediante una clase.
Un atributo estática de una clase es un dato común a todas las instancias de esta clase y accesible en todos los métodos. Un atributo estático es incluso accesible sin instancia alguna.
En programación orientada a objetos, una variable de clase o miembro de dato estático es una variable, al contrario que las variables de instancia, propias de la clase que la contiene y no de instancias de la misma. Esto quiere decir que todos los objetos que se creen de esta clase comparten su valor, son llamadas variables estáticas.

En el código siguiente definimos una clase con una variable miembro estática:

```
1.  class MiClase {
2.  public:
3.      static int mi_variable;
4.  };
5.  // Esta variable se inicializa así:
6.  int MiClase::mi_variable = 5;
7.  // Si la queremos usar sin instancia, entonces usamos el nombre cualificado:
8.  void haz_algo(void) {
9.      MiClase::miVariable = 3;
10. }
```

CONST (3) (2/2)

http://www.zator.com/Cpp/E3_2_1c.htm
http://c.conclase.net/curso/index.php/?cap=033

(3)

¿Qué significa la palabra **const** cuando **sucede** a una declaración de un **método de una clase**? **Ejemplifique con una clase y una invocación donde sea necesario su uso.**

Explique qué **importancia** tiene el uso de **const** al momento de **crear una clase** que será publicada para el uso por parte de otros programadores.

Si la palabra **const** sucede en la declaración de un método de una clase, se asegura que no va a ser modificado el estado interno del objeto (es decir sus atributos).
Si el método no cumple con la premisa de mantener intacto el estado interno del objeto entonces el programa no va a compilar.
Es una práctica muy util para los programadores, ya que asegura que no se esté modificando el estado interno del objeto por error.
Para asegurar su premisa, los métodos const solo pueden llamar a otros métodos const, de ésta manera se asegura que los métodos llamados por el método tampoco modifiquen el estado interno del objeto.

```
1.  class ClaseA {
2.  private:
3.      int value;
4.  public:
5.      ClaseA(int i) {value = i;}
6.      //Se declaró getValue() como un método que no cambia el estado interno del objeto.
7.      //Falla en la compilación porque se modificó el atributo value.
8.      //Le permite detectar el error al programador en la etapa de compilación.
9.      //Sin el const este error NO se hubiese detectado en la etapa de compilación.
10.     int getValue() const {
11.         value++; // <- Error del programador detectado por usar const
12.         return value;
13.     }
14. };
```

¿Qué significa el prefijo **const** antepuesto a un **parámetro** de una función? Ejemplifique.

El prefijo const antepuesto a un parámetro de una función/método indica que el mismo (el parametró) no será modificado o reasignado.
Ejemplo simple:

```
1.  void foo(const int &i) { //No compila porque se modifica el estado del parámetro
2.      i++;
3.  }
4.
5.  int main(){
6.      int a = 10;
7.      foo(a);
8.      return 0;
9.  }
```

Ejemplo más complejo:

```
1.  class A {
2.  public:
3.      void metodoConstante() const {
4.          i++; //NO compila xq modifica el estado interno del objeto.
5.      }
6.      void metodo() {
7.          i++;
8.      }
9.  private:
10.     int i;
11. };
12.
13. void foo(const A &a) {
14.     //Compila siempre y cuando el método internamente no modifique el estado interno del objeto.
15.     a.metodoConstante();
16.
17.     //NO compila xq si '&a' es constante solo puede llamar a métodos constantes
18.     a.metodo();
19. }
20.
21. int main(){
22.     A a;
23.     foo(a);
24.     a.metodo(); //Compila porque 'a' no es constante -> puede llamar a metodo() a pesar de que éste no sea constante.
25.     return 0;
26. }
```

Manejo Entrada/Salida en C y C++ (5) (2/2)

(3)
¿Qué objeto se provee en ISO **C** para manejar la **consola de entrada**? **Ejemplifique**. (Faltaria un ejemplo de salida)
stdio.h es el archivo de cabecera que contiene las definiciones de las macros, las constantes, las declaraciones de funciones de la biblioteca estándar del lenguaje de programación C para hacer operaciones, estándar, de entrada y salida.
El objeto que se provee se llama 'stdin' y es un File pointer al buffer de entrada estandar.

```c
#include <stdio.h>
int main() {
    int c = fgetc(stdin); // obtengo un byte desde stdin.
    printf("El byte ingresado: %c\n", c);
    return 0;
}
```

(2) ¿Qué Objetos se provee en forma standard en **C++** para el manejo de la **entrada y salida standard**?. **Ejemplifique su uso**.
El conjunto de funciones y clases relacionadas a operaciones de entrada y salida se incluyen en los archivos de la cabecera iostream.h.
Por lo que debemos agregarlas con la sentencia: #include <iostream.h>
Los objetos de flujo que vienen predefinidos serán:

- cin, que toma caracteres de la entrada estándar (teclado);
- cout, pone caracteres en la salida estándar (consola/terminal);
- cerr y clog ponen mensajes de error en la salida estándar.

```c
#include <iostream>
int main() {
    int i;
    std::cout << "Introduce un número: ";
    std::cin >> i;
    std::cout << "El número ingresado es " << i << std::endl;
    return 0;
}
```

Otros (24) (6/11)

¿Qué significa this en C++? ¿Dónde se usa explícita o implícitamente? ¿Dónde no es necesario?
Es un puntero que mantiene la dirección de memoria del objeto actual, es decir, aquél usado para llamar al método.
Se usa explícitamente para diferenciar ambigüedades cuando aparecen variables o parámetros con el mismo nombre que algún atributo de la clase (i) o para retornar una referencia al mismo (ii), e implícitamente al usar atributos del objeto (iii)
No es necesario cuando no tenemos ningún parámetro del método que se llame igual a un atributo de clase, es decir cuando no se genere ambiguedad. (Caso iii)

Ejemplos:

```c
class A {
public:
    A(int valor);
    A* getSelf();
    void metodoCualquiera();
private:
    int valor;
};

// (i) Expliciatamente para diferenciar ambiguedades entre un parametro y un atributo de clase.
A::A(int valor) {
    this->valor = valor;
}

// (ii) Para retornar una referencia de si mismo
A* A::getSelf() {
    return this;
}

// (iii) Implicitamente
void A::metodoCualquiera() {
    valor = 10; //equivalente a this->valor = 10;
}
```

(3) ¿Qué es un **parámetro opcional** 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. En c++ solamente pueden ser opcionales los últimos parámetros.
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:

```c
//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
```

(3) Explique el concepto de referencia en C++. ¿Qué diferencias posee con el uso de punteros? **Ejemplifique.**

En C todo se pasa por copia. Si queremos pasar por referencia en realidad lo que hacemos es pasar por copia un puntero.
En C++ el pasaje de parámetros por referencia significa que se recibe un alias de lo que se haya pasado por parámetro. Es decir que toda modificación aplicada a dicha referencia se verá reflejada en la instancia pasada por parámetro.

Pasaje Puntero por valor	**vs.**	**Pasaje por Referencia**
Ocupa espacio en memoria sizeof(int)		No ocupa espacio en memoria (depende del compilador)
Se debe desreferenciar con *		No es necesario desreferenciar (facilita programación)
Puede cambiar el objeto al que apunta		Una vez inicializada la referencia a un objeto no se puede hacer que referencie a otro

```
1.  #include <iostream>
2.  void incEntero(int &i) {
3.      i++;
4.  }
5.
6.  int main() {
7.      int a = 0;
8.      std::cout << a << std::endl; //>>> a = 0
9.      incEntero(a);
10.     std::cout << a << std::endl; //>>> a = 1
11.     return 0;
12. }
```

(3) ¿Qué diferencia existe entre el **pasaje de parámetros por valor** (ej.: void funcion(Clase O);) y el pasaje **por referencia constante** (ej.: void funcion(const Clase &O);)?

En el pasaje de parámetros por valor se copia el objeto que se pasa por parametro (consume más memoria) y los cambios aplicados afectarán al objeto copia.
En el pasaje por referencia se pasa una referencia del objeto (lo que permite evitar duplicar el objeto) pero los cambios aplicados afectarán al objeto se que pasó por referencia.
Si queremos el beneficio del pasaje por referencia (evitar duplicar memoria) pero al mismo tiempo queremos el beneficio de que no se altere el objeto pasado por parametro como sucede en el pasaje por valor se puede utilizar un pasaje por referencia constante para asegurarnos que el objeto que se pasa por referencia no sea alterado ni modificado.

(3)

¿Qué es un **método virtual**? ¿Para qué se usan? **Ejemplifique**.

Distinguir entre virtual y no virtual sirve para resolver el problema de la ambiguedad 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.

```
1.  #include <iostream>
2.  class Animal {
3.  public:
4.      virtual void come() { std::cout << "Yo como como un animal genérico.\n"; }
5.      virtual ~Animal() {}
6.  };
7.
8.  class Lobo : public Animal {
9.  public:
10.     void come() { std::cout << "¡Yo como como un lobo!\n"; }
11.     virtual ~Lobo() {}
12. };
13.
14. class Pez : public Animal {
15. public:
16.     void come() { std::cout << "¡Yo como como un pez!\n"; }
17.     virtual ~Pez() {}
18. };
19.
20. class OtroAnimal : public Animal {
21.     virtual ~OtroAnimal() {}
22. };
23.
24. int main() {
25.     Animal *unAnimal[4];
26.     unAnimal[0] = new Animal();
27.     unAnimal[1] = new Lobo();
28.     unAnimal[2] = new Pez();
29.     unAnimal[3] = new OtroAnimal();
30.
31.     for(int i = 0; i < 4; i++)
32.         unAnimal[i]->come();
33.     for (int i = 0; i < 4; i++)
34.         delete unAnimal[i];
35.
36.     return 0;
37. }
38.
```

Salida **con** el método virtual come:

```
Yo como como un animal genérico.
¡Yo como como un lobo!
¡Yo como como un pez!
Yo como como un animal genérico.
```

Salida **sin** el método virtual come:

```
Yo como como un animal genérico.
Yo como como un animal genérico.
Yo como como un animal genérico.
Yo como como un animal genérico.
```

(2) ¿Qué es una función **virtual pura**? ¿Para qué se utiliza?¿Cómo se declara?

Un método virtual puro es un método que no está implementado en la super clase, por lo cual las subclases deben redefinir e implementar el método obligatoriamente.
Si una clase tiene al menos un método virtual puro dicha clase será una clase abstracta, y por tanto la misma no puede ser instanciada.
Declaración de método virtual puro:

```
class Animal {
public:
    virtual void come()=0 //Declaramos el método virtual puro con =0
};
```

(3) ¿Para qué se utilizan los **destructores virtuales**? ¿Cuando son **necesarios**? **Ejemplifique**.

Los destructores virtuales se utilizan en las claseas padre para que primero se llame al destructor de la clase derivada, para luego ejecutarse el destructor de la clase padre.

Son necesarios por ejemplo al momento de escribir una clase que usa memoria dinámica y que es usada con polimorfismo.
Tendría la precaución de hacer que el destructor de la clase padre sea un método virtual.
Cuando se instancia b1 en memoria dinamica con el new, el objeto es de tipo B, pero el puntero es de tipo A. Por lo tanto, cuando se realice el delete b1 se llamará al destructor de la clase A ya que vemos al objeto como de tipo A.
Si el ~A() no es virtual entonces nunca se llamará al ~B(), con lo cual no se liberará la memoria asociada a la clase B.
Si el destructor de A es virtual (virtual ~A()) entonces primero se llama al destructor de la clase derivada, es decir a ~B() liberando los recursos de B, y luego de ejecutarse el destructor de B se ejecuta el destructor de A.
Caso delete b2:
Como el puntero es de tipo B, se llama directamenteal destructor de B el cual a su vez llamará luego al destructor de la clase padre A.

Ejemplo:

```
24.  class A {
25.  protected :
26.      int x;
27.  public :
28.      int y;
29.  public :
30.      virtual ~A() {}
31.  };
32.
33.  class B : public A {
34.  private :
35.      int z;
36.  public :
37.      ~B() {}
38.  };
39.
40.  int main() {
41.      A* b1 = new B(); // Instancio b1 de tipo B, pero el puntero es de tipo A (padre de B).
42.      delete b1; //Llama a ~A(), ve que es virtual -> llama al destructor de la derivada ~B() y luego llama al destructor del padre ~A().
43.      B* b2 = new B(); // Instancio b2 de tipo B, y el puntero es de tipo B.
44.      delete b2; //Llama a ~B() y luego llama al destructor del padre ~A().
45.      return 0;
46.  }
```

Enumere y explique con claridad los **tipos de accesos** a métodos y atributos de las clases, existentes en C++**.**

(2) ¿Qué **son los clases abstractas**? ¿Para qué sirven? **Escriba declaraciones** de métodos/clases que ejemplifiquen sus conceptos.

Lenguaje C/C++ (Implementación)

Cambios de base (9) (0/6)

(3) Escriba una aplicación C++ que lea de consola un número en base 6 y una base (entre 4 y 16), e imprima el número leído en la base especificada.

(2)
Escribir un programa C que reciba 3 números (a, b y c) por línea de comandos e imprima, en octal y con forma de tabla los siguientes valores:
a/2*b, a/(2*b+1), a/(2*b+2)...a*c.
Considere valores negativos, con decimales. Valide posibles errores.
<simil>
Escribir un programa ANSI C que reciba 3 números (a, b y c) por línea de comandos e imprima, en hexadecimal y con forma de tabla los siguientes valores: a/b, a/(b+1)...a*c.
Considere valores negativos, con decimales. Valide posibles errores.

Escriba una aplicación C que lea de la línea de comandos un número y lo imprima en todas las bases entre 2 y 16.

Implemente la función unsigned BaseLoad(char *S, unsigned short B) que procese la secuencia de símbolos en base B de la cadena S, devolviendo la cifra representada como resultado de la función.

Escriba una aplicación ANSI C que tome dos nros. octales por línea de comandos e imprima su multiplicación en base hexadecimal.

Implemente la función void Print(int I, int Base) que imprima el valor I en base Base.

Numeros (5) (0/5)

Escribir un programa C que admita por línea de comandos 2 números e imprima la división entre ambos. Considere errores.

Escribir un programa C no recursivo que reciba una CANTIDAD (entero) por línea de comandos. El programa debe imprimir CANTIDAD términos de una serie en la cual los primeros 3 elementos son 1 y los siguientes se calculan como la suma de los 3 términos anteriores.

Escriba un programa C que tome por línea de comandos 2 parámetros: N y K e imprima por stdout los K términos de la serie de Fibonacci comenzando por el término N (es decir: Fn, Fn+1,..., Fn+k-1) . Nota: F0=1,F1=1,F2=2.

Escribir un programa ANSI C que reciba 3 números (a, b y c) por línea de comandos e imprima, en forma de tabla, a*b, a*(b+1)...a*c.

Escriba un programa C que reciba por línea de comandos los valores a, b y L y escriba la serie Sn mientras su valor sea inferior a L:

S1=6
S2=2
S3=11
Sn=a*Sn-2+b*Sn-1-Sn-3 (n>3)

Cabe destacar que los parámetros serán ingresados en forma de texto con coma (no punto) decimal.

Cadenas (16) (2/14)

C

1) **(2)** Escriba un programa C que reciba como parámetros 2 cadenas (A y B) e imprima por stdout la cadena A sin las ocurrencias de B. No utilice ninguna función de librería.

2) Escriba un programa C que tome 3 cadenas por línea de comandos: A, B y C; e imprima la cadena A después de haber reemplazado todas las ocurrencias de B por C.
ej.: reemp.exe “El ejercicio está mal hecho” mal bien -----> El ejercicio está bien hecho

3) Escriba una aplicación C que reciba una frase de la línea de comandos (en 1 sólo segmento) e imprima:
a) cantidad de consonantes;
b) cantidad de palabras;
c) cantidad de palabras con, al menos, 3 vocales distintas.

Escriba un programa C que tome por línea de comandos 2 parámetros: (X1,Y1) e imprima por stdout la ecuación de la recta que pasa por (3,5) y por el punto especificado.

Escriba una funcion 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.

C++

(2)

Escriba una aplicación C++ que acepte un único parámetro por línea de comandos (con una oración) e imprima las palabras de la oración ordenadas descendentemente de acuerdo a su cantidad de vocales.

<simil>

Escriba una aplicación C++ que acepte un único parámetro por línea de comandos (con una oración) e imprima las palabras de la oración ordenadas ascendentemente de acuerdo a su cantidad de letras.

Escriba una aplicación C++ que acepte un único parámetro por línea de comandos (con una oración) e imprima las palabras de la oración ordenadas alfabéticamente.

Escribir un programa que permita ingresar 3 cadenas desde STDIN e imprima la primera, reemplazando las ocurrencias de la segunda por la tercera.

Escriba una aplicación ANSI C++ que tome dos strings por línea de comandos e imprima un tercer string resultado de intercalar las letras de ambos string de acuerdo a su precedencia alfabética.

Escriba un programa que efectúe reemplazos en un texto. El programa será incovado como REEMPLAZAR.EXE <SIMBOLO> <REEMPLAZO>. Deberá leer un texto desde stdin y escribirlo en stdout, reemplazando todas las ocurrencias de <SIMBOLO> por <REEMPLAZO>, excepto que <SIMBOLO> se encuentre entre comillas (dentro de una cadena). Considere que ni las cadenas ni <SIMBOLO> nuncá aparecerán partidos en más de un renglón.

Definir la rutina PRIMERO_PRIMOS de forma que ordene los elementos recibidos de la siguiente manera: Primero los primos y luego el resto, y dentro de esa clasificación, por orden ascendente. Ejemplo: Si recibe los elementos 1,6,7,2,9,4,3 debe devolver 1,3,7,2,4,6,9
std:list<T> PRIMERO_PRIMOS(std:list<T> &Lista);
Nota: Asuma que el tipo T posee el operador < y el método bool EsDivisible(int Div).

Implemente la función void String_a_Unsigned(char *s, unsigned *b) que interprete la cadena s (de 32 1s/0s) y guarde el valor correspondiente en el entero sin signo indicado por b.

Escriba un programa que reciba por línea de comandos los coeficientes (a, b, c) del polinomio aX2+bX+c y lo grafique (en modo texto) para el intervalor [-20-20]. Se debe tener en cuenta que deben graficarse los ejes. El eje X puede tener escala fija, pero el eje Y debe escalarse de acuerdo a los valores del polinomio (para que siempre se pueda visualizar).

Otros (10) (2/5)

¿Qué es un iterador?¿Cómo se utiliza?

Un iterador es un objeto que permite al programador recorrer una colección sin que sea necesario que conozca la implementación de la misma.
Los contenedores STL se pueden recorrer utilizando iteradores, por ejemplo

```
for( auto it = contenedor.begin(); it != contenedor.end(); ++it ) {
   // ...
}
```

(2)

Suponiendo que el puntero P apunta a la dirección de memoria 500, ¿A qué dirección apunta cada una de las siguientes expresiones:

1. ((unsigned int *) P)+1
2. ((float *) P)+2
3. ((short int *) P)+3

Escriba una nota donde aclare las suposiciones de plataforma asumidas en la respuesta.

Asumo que las direccion de memoria está dada en base 10.
Respuesta para una arquitectura de 32bits:
1- 500+1*4 = 504
2- 500+2*4 = 508
3- 500+3*2 = 506
Respuesta para una arquitectura de 64bits:
1- 500+1*8 = 508
2- 500+2*8 = 516
3- 500+3*4 = 512

<simil>

Suponiendo que el puntero P apunta a la dirección de memoria 1000, ¿A qué dirección apunta cada una de las siguientes expresiones:

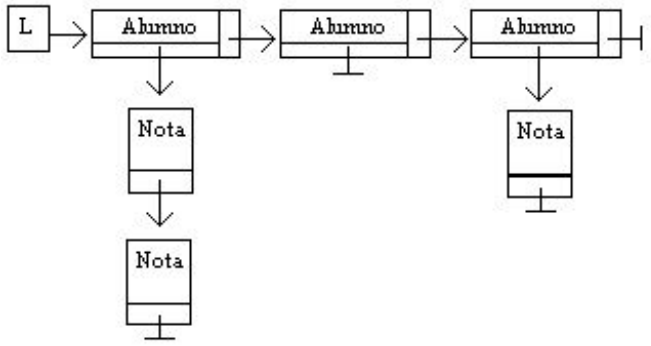
1. ((int *) P)+1
2. ((char *) P)+2
3. ((float *) P)[3]

Nota: Aclare las suposiciones de plataforma asumidas en la respuesta.

Asumo que las direccion de memoria está dada en base 10.
Respuesta para una arquitectura de 32bits:
1- 1000+1*4 = 1004
2- 1000+2*1 = 1004
3- 1000+3*4 = 1012
Respuesta para una arquitectura de 64bits:
1- 1000+1*8 = 1004
2- 1000+2*1 = 1004
3- 1000+3*8 = 1012

```
//Con este código puedo verificar como se desplaza el puntero (salida con direcciones hexadecimales).
#include <stdio.h>
#include <stdlib.h>
void main() {
   int A;
   int* ptrA = &A;
   void* ptrA2 = ptrA+1; //ptrA es *int entonces mueve 1*4bytes
   void* ptrA3 = ((int *) ptrA)+1; //ptrA es *int, se castea a *int entonces mueve 1*4bytes
   void* ptrA4 = ((char *) ptrA)+1; //ptrA es *int, se castea a *char entonces mueve 1*1byte
   printf("&A\t= %p\n", (void*)&A);          //Ej: 1000
   printf("ptrA\t= %p\n", (void*)ptrA);      //Ej: 1000
   printf("ptrA2\t= %p\n", (void*)ptrA2);    //Ej: 1004
   printf("ptrA3\t= %p\n", (void*)ptrA3);    //Ej: 1004
   printf("ptrA4\t= %p\n", (void*)ptrA4);    //Ej: 1001
}
```

Implemente la función Liberar que libere la siguiente estructura de memoria:




```
struct Alumno {
  int Padron;
  char *Nombre;
  struct Alumno *Sig;
  struct Nota *Notas;
};

struct Nota {
  float Calificacion;
  struct Nota *Sig;
};
```

void Liberar(struct Alumno *L);

Indicar la salida del siguiente programa:

```
#include <iostream>
class Base {
public:
    static void f1 (void) { std::cout << "B.f1" << std::endl; }
    virtual void f2 (void) { std::cout << "B.f2" << std::endl; f1(); }
    virtual void f3 (void) { std::cout << "B.f3" << std::endl; f2(); f1(); }
};

class Derivada : public Base {
public:
    static void f1 (void) { std::cout << "D.f1" << std::endl; }
    void f2 (void) { std::cout << "D.f2" << std::endl; f1(); }
    void f3 (void) { std::cout << "D.f3" << std::endl; f2(); f1(); }
};

int main() {
    Derivada D;
    Base* pB = &D;
    Derivada* pD = &D;
    pB->f1(); //B.f1
    pB->f2(); //D.f2 D.f1
    pB->f3(); //D.f3 D.f2 D.f1 D.f1
    pD->f1(); //D.f1
    pD->f2(); //D.f2 D.f1
    pD->f3(); //D.f3 D.f2 D.f1 D.f1
    return 0;
```

Se indicó la salida comentada al lado de cada línea.
Si el método no es virtual lo va a buscar en la clase del tipo con el que se OBSERVA al objeto.
Si el método es virtual va a buscar el método primero en el tipo real del objeto y si no lo encuentra en la clase Padre.
Si el método es virtual puro va a buscar el método definido en las clases hijas.

Base* pB = &D; //En este caso Base es el tipo con el que se OBSERVA al objeto y Derivada es el tipo REAL del objeto.
Derivada* pD = &D; //En este caso Derivada es el tipo con el que se OBSERVA al objeto y Derivada es el tipo REAL del objeto.

(3)
Escriba un pequeño programa que reciba por línea de comandos un nombre de archivo, una dirección hexadecinal y una cantidad de bytes N; e imprima los N bytes (del archivo) existentes a partir de la dirección indicada, en el siguiente formato:
xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx
xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx
<simil>
Escriba un pequeño programa que reciba por línea de comandos una dirección hexadecinal de memoria y una cantidad de bytes N; e imprima los N bytes de memoria existentes a partir de la dirección indicada, en el siguiente formato:
xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx | cccccccc cccccccc
xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx | cccccccc cccccccc
<simil>
Escriba un pequeño programa que imprima el contenido de un archivo cuyo nombre se recibe por línea de comandos, con la forma:
dddddddd xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx | cccccccc cccccccc
dddddddd xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx | cccccccc cccccccc

(3)
¿Qué es necesario que tenga la clase EJEMPLO para que sea válido el siguiente código?
LaClase A(1);
LaClase B(3.5);
A=(A= =B)? (MiClase() : MiClase(“2.1”));
Justifique.
<simil>
¿Qué es necesario que tenga la clase EJEMPLO para que sea válido el siguiente código?
MiClase C[2];
C[1]=(C[0]>C[1])? (MiClase(7.3) : MiClase(“7.2”));
Justifique.
<simil>
¿Qué es necesario que tenga la clase EJEMPLO para que sea válido el siguiente código?
EJEMPLO A[2];
A[1]=((float)A[0]==(int)A[1])?EJEMPLO(1.5): EJEMPLO(“3.2”);
Justifique.

Excepciones (1/1)

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

Las excepciones son mecanísmos para manejar circunstancias excepcionales (por ejemplo errores en tiempo de ejecución).
Cuando una circunstancia excepcional surge, el bloque que la detecta lanza/arroja (**throw**) una excepción.
La excepción va escalando hasta ser capturada por el bloque (**catch**), el cual puede o relanzarla o manejarla.
El objetivo es que cuando se produzca una situación excepcional se proceda de forma ordenada para manejarla. Bien sea para continuar la ejecución, para mostrar el mensaje de error o para salir de forma ordenada.

```
1.  #include <iostream>
2.
3.  float dividir(const float dividendo, const float divisor) {
4.      if (divisor == 0)
5.          throw std::runtime_error("ERROR: No se puede dividir por 0.");
6.      else
7.          return dividendo/divisor;
8.  }
9.
10. int main () {
11.     try {
12.         std::cout << dividir(3, 4) << std::endl;
13.         std::cout << dividir(3, 0) << std::endl; //Lanza excepción
14.     }
15.     catch (const std::runtime_error &e) { //Capturo excepción
16.         std::cerr << e.what() << std::endl;
17.     }
18.     return 0;
19. }
```