1 Preguntas Teoricas

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

   La distincion entre metodos virtuales y no virtuales permite determinar en que momento se va resolver cual funcion utilizar. Caso el metodo no sea virtual, se define cual funcion utilizar en tiempo de compilacion (static linkage), mientras que si la funcion es virtual, se define en runtime, tambien conocido como dynamic linkage.

   Si una funcion tiene el atributo virtual y tenemos un caso de una ambiguedad, como por ejemplo un puntero del tipo *ClaseBase* que apunta a alguna clase derivada, se llamara a la definicion del metodo de la clase derivada. Caso contrario, si no hacemos el *virtual* de forma explicito, se llama al de la clase padre.

2. **Describa con exactitud cada uno de los siguientes:**

   (a) **static int A = 7**; Se declara una variable del tipo entero, con valor 7, que tendra duracion durante todo el programa, es decir, vivirá en el data segment. Su visibilidad sera local. Puede ser en todo el archivo o dentro de una funcion, dependiendo de donde se la declare.

   (b) **extern char* (*B) [7]**; No se asigname memoria. Apenas se setea un nombre para un vector de 7 posiciones de punteros a char.

   (c) **float **C** Se declara una variable con el nombre C en que guardara el valor de un puntero que apunta a un float.

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

   Si tenemos dos threads tratando de accerder apenas para lectura, no existen race conditions o comportamientos indeterminados, ya que nunca se modifica el valor. Si al menos uno de los threads va a modificar el valor (escritura), es necesario transformar la operacion de lectura o escritura en una operacion atomica. Esto se puede realizar mediante el uso de un mutex, que impide que los dos hilos accedan simultaneamente al valor.

4. **¿Qué valor arroja sizeof(int)? Justifique**

   Depende de la arquitectura y del compilador. Si estamos trabajando en una arquitectura de 32 o 64 bits, generalmente obtendremos el valor de 4 (bytes). En una arquitectura de 16 bits, un int ocupa generalmente 2 bytes.

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

   Una funcion es bloqueante cuando el hilo que la esta ejecutando se "frena" en esa funcion hasta obtener un resultado (de un medio externo). Una forma de mantener el programa "vivo" es mediante la utilización de threads que se encarguen de esperar el resultado de la función, mientras que el otro hilo continua con la ejecucion.

6. **Explique qué es y para qué sirve un constructor de copia en C++:**

   (a) **Indique cómo se comporta el sistema si éste no es definido por el desarrollador**
   Si el constructor por moviemiento no esta definido, se hacer una copia "default", esto es, una copia bit a bit del elemento.

   (b) **Explique al menos una estrategia para evitar que una clase sea cópiable**
   Si queremeos que una clase no sea copiable podemos definir al constructor por copia y al operador asignacion como "delete", es decir, como eliminados. De esta forma si tratamos de realizar una copia, el error sera detectado en tiempo de compilacion.

   Ej:

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

(c) **Indique qué diferencia existe entre un constructor de copia y uno move**
Un constructor por copia, como el propio nombre dice, no deja invalido al objeto que se esta copiando, si no que crea un nuevo objeto con los mismos valores que el anterior. El contructor por moviemiento le cede el ownership del objeto originial al nuevo objeto, dejando el objeto original en un estado invalido, que ya no deberia usarse mas.

7. **Describa con exactitud cada uno de los siguientes:**

(a) **void (*F)(int i)**
Se declara a F como una variable que apunta a una funcion que recibe un entero y no devuelve nada (void).

(b) **static void B(float a, float b){};**
Se declara una funcion que recibe dos float y no devuelve nada. Su validez es local en todo el archivo, no tiene validez afuera.

(c) **int *(*C)[5];**
Se declara a C como una arregle de punteros a punteros a enteros de tamaño 5.

8. **¿Qué es una macro de C? Describa las buenas prácticas para su definición y ejemplifique**
Una macro es una operacion definida mediante instrucciones al compilador que luego seran expandidas (es decir, remplazar donde dice su nombre por el codigo) por el propio compilador en la etapa de precompilacion.
Ej:

```
#define CUBE(A) (A)*(A)*(A)
```

9. **Describa el proceso de transformación de código fuente a un ejecutable. Precise las etapas, las tareas desarrolladas y los tipos de error generados en cada una de ellas.**
El proceso de compilacion consta de tres etapas principales:

- *Precompilacion:* En esta etapa el compilador expande las macros, remplaza las constantes, interpreta las directivas del compilador en general.
- *Compilacion:* En esta etapa se realiza un analisis sintactico, semantico generando un codigo objecto.
- *Linkeo:* En esta etapa el linker resuelve las referencias externas, generando un unico codigo binario ejecutable.

10. **¿Qué ventaja ofrece un lock raii frente al tradicional lock/unlock?**
Una de las principales ventajas es la imposibilidad de dejar el lock activado, sea tanto por olvidarse o por que se lanza alguna excepcion durante la ejecucion del codigo que encierra el mutex. De esta manera, podemos asegurarnos que siempre al salir del scope de la funcion, vamos a poder acceder desde otro thread al objeto protegido.

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

(a) **Una variable global static** Se inicializa al comienzo del programa y va a vivr hasta que este termine. Vive en el data segment. Si no se inicializa de forma explicita comienza con el valor 0 (puede variar con el lenguaje). Su visibilidad es global dentro del archivo.

(b) **Una variable local** Se inicializa al llamar una funcion que la contiene y su tiempo de vida finaliza cuando se sale del scope de la funcion. Vive en el stack. Su visibilidad es apenas dentro de la funcion y no son inicializadas por defecto.

(c) **Un atributo de clase static** Se inicializa al comenzar el programa. Al igual que una variable static comun, se inicializa con 0 si no se lo hace de forma explicita. El valor del atributo se comparte entre todas las instancias de la clase y en caso de declararse público es accesible desde fuera de la clase, anteponiendo el nombre de la misma (en lugar una instancia es Clase::Attrib). Se puede acceder desde cualquier archivo que importe la clase si se declara publica.

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

    Es una forma de decidir que bloque de codigo se debe compilar en base a una condicion. Esto se resuelve en la etapa de precompilacion.

    Ej:

```
#ifdef PI
    printf("%f", PI);
#else
    printf("3.1416");
#endif
```

    Puede ser util cuando se queire escribir codigo portable o agregar codigo para depuracion.

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

    Porque los templates no son compilados, por ende, no existen los archivos objetos. El codigo de un template es generado por el compilador durante el preprocesamiento, en base al tipo de dato que se utilizo en el template, y se genera uno distinto para cada tipo de dato.

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

    La palabra virtual en un metodo de clase permite lo que se conoce como dynamic binding, es decir, se decir cual es el metodo a utilizar, si el de la clase padre o alguno de sus hijas, en tiempo de ejecucion y no de compilacion. De esta forma, permite que redefinamos metodos en las clases derivadas y trabajar de modo polimorfico. En el código objeto esto genera una diferencia debido a que el dynamic binding requiere tener una tabla de métodos virtuales para cada objeto (la VTable) de modo de permitir determinar en tiempo de ejecución qué método debe ser llamado.

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

    Porque el codigo se va a generar despues para cada tipo de dato. Es decir, el codigo objeto no es unico, se compila un codigo diferente para cada tipo utilizado en el template. Por lo tanto, el compilador necesita tener acceso a la implementación de la clase y sus métodos antes de encontratosr estos argumen para instanciarlos con el argumento del template.

16. **¿Qué es un deadlock? Ejemplifique**

    Un deadlock es un error que se da en tiempo de ejecucion en que dos o mas hilos se quedan bloqueados sin poder salir debido a que un proceso evita la condicion del otro e viceversa.

```
#include <mutex>
#include <thread>
std::mutex mtx;

void funcio() {
    mtx.lock()
    /*hago algo*/
    mtx.unlock();
}

int main() {
    mtx.lock();
    std::thread hilo(funcion);
    hilo.join();
    mtx.unlock();
    return 0;
}
```

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

    Una variable de clase es una variable que se comparte por todos los objetos de una determinada clase, es decir, al modificar el valor en alguno de los objetos, este se modifica para todos.

    Ej:

```
#include <iostream>

class Clase {
    public:
        static int numero;

        Clase(): numero(1) {};
};

int main() {
    std::cout << Clase::numero << "\n";
    return 0;
}
```

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

    Es un tipo de constructor en que no se duplican los valores del objeto movido, si no que se transfiere los valores dejando al objeto inicial en un estado invalido. Esto es mucho mas eficientes que las copias ademas de ser muy util cuado se tiene objetos que no tiene sentido la copia, como un Socket, que no tiene sentido la copia del file descriptor. Si no se tienen el contructor por moviemiento, no se va a generar uno defualt, si no que se untiliza la asignacion por copia o el constructor por copía default o no caso este definido.

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

    f es un puntero a funcion que recibe un puntero a flaot y un array de tres enteros sin signo y devuelve un char.

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

    Durante el proceso de compilacion se transforma el codigo en un codigo objeto. En esta etapa se realizan los analisis sintacticos y semanticos. En la etapa de linkeo se resuelven las referencias externas a otros modulos, generando un unico codigo binario ejecutable.

```
int sumar(int num1, int num2);

int main() {
    numero_t num;
    return 0;
}
```

    Error de compilacion: numero_t es un tipo de dato no especificado, entonces no se puede compilar ya que no se sabe cuanto espacio en el stack se deberia reservar.

    Error de linkeo: La funcion sumar esta declarada, por lo cual se pudo compilar pero no esta definida. El linker falla en encontrar la referencia adonde esta definida la funcion.

21. **Indique la salida del siguiente programa:**

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

    ~A() {
        cout << \~A()" << endl;
    }
};

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

    ~B() {
        cout << \~B()" << endl;
    }
}

int main() {
    Bb;
    return 0;
}
```

La salida es:

A() B() B() A()

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

Un mutex es un objeto que significa "mutual exclusion", es decir, evita que dos o mas threads accedan de forma simultanea a un atributo, evitando asi las race condition. Si un hilo esta accediendo a un atributo y otro hilo trata de acceder, eeste bloquea al segundo hilo hasta que el hilo que esta modificando/leyendo la variable termine.

Es necesario cuando necesitamos realizar operacion de forma atomica. Ej:

```
#include <mutex>
#include <thread>
std::mutex mtx;

void sumar_si_menor(int* numero, int restriccion) {
    mtx.lock();
    if (*numero < restriccion)
        (*numero)++:
    mtx.unlock();
}

int main() {
    int numero = 1, restriccion = 2;

    std::thread hilo(sumar_si_menor, &numero, restriccion);
    sumar_si_menor(&numero, restricccion);
    hilo.join();
    return 0;
}
```

Si el codigo anterior se ejecutarar sin un mutex, se puede dar el caso en el que numero termine en 3, debido a una race condition.

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

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

    (a) Su propio PC
    (b) Su propio conjunto de registros
    (c) Su propio stack
    (d) Espacio de direcciones compartido

    Para lanzar un thread en C++ aparte del principal podemos hacer:

    `std::thread nombre_hilo(func, arg_1, arg_2, ... arg_n)`

    Para esperar un thread existe el metodo join, que espera que termine su ejecucion y luego lo une al hilo principal.

24. **Dentro del siguiente código:**

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

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

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

    La funcion listen le avisa al SO que el socket previamente bindeado a una direccion especifica desea comenzar a escuchar conexiones. Sus parametros son el file descriptor del socket aceptador y la longitud maxima de la cola de espera de conexciones para ser aceptadas.

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

    Un uso puede ser cuando querramos decir que una funcion no va a modificar un valor. Permite un chequeo en tiempo de compilacion evitando asi un bug durante el runtime.

```
size_t largo\_string(const char* string) {
    return strlen(string);
}
```

    Si se tratara de moficar al string dentro de la funcion, podemos detectar el error en tiempo de compilacion.

    Otro uso es cuando queremos decir que un metodo de una clase nmo modifica el estado interno.

```
class Container {
    std::string cadena;

    std::string& get_cadena() const {
        return this->cadena
    }
}
```

Al llamar al metodo get_cadena estamos seguros que no se va a modificar el estado interno de nuestra clase.

27. **Escriba:**

    (a) **La definición de una función de alcance local que toma como parámetros un puntero a entero con signo y un puntero a carácter con signo; sin retorno.**
    static void f (unsigned*, char*)

    (b) **La declaración de un puntero a función que respete la firma de la función anterior.**
    static void (*f) (unsigned*, char*);

    (c) **La declaración de una variable global entera que se encuentra en otra unidad de compilación.**
    extern int a; Wikipedia dice: "external variables are globally accessible"

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

    Los usos de static en C++ son 4:

    - Variable static en funciones:
      Son variables que se inicializan una sola vez y quedan modificadas durante todo el programa, es decir, cada vez que llamo a la funcion, recuerda el valor anterior. Se guardan en el data segment. Una vez que se define un miembro de clase static no se puede reedefinir pero se le pueden aplicar operaciones aritméticas.
    - Objetos de clase static
      Se guardan en el data segment y tiene como lifetime hasta el final del programa. Usan los constructores de manera normal y no tienen un valor por default en caso de ser una clase definida por el usuario. Se destruyen cuando sale el scope de main y no del scope de creación,
    - Miembros de clase static
      Son atributos que todas la clases comparten con el mismo valor. Modificarlo en una clase implcia que en todos los objetos de esa clase tambien se modifique. Este valor esta guardado una unica vez en el data segment.
    - Métodos de clase static
      Son metodos que no pueden acceder a las variables internas de una clase. solamente a los miembros de clase estáticos y funciones estáticas. No se puede usar el operador this.

29. **Escriba las siguientes definiciones/declaraciones:**

    (a) **Una declaración de una función llamada F que tome un arreglo de flotantes y devuelva un puntero a un arreglo de enteros cortos sin signo.**
    unsigned short []* F(float A[]);

    (b) **Una definición de una variable entero largo con signo, global, visible fuera del fuente, llamada G.**
    extern long int G;

    (c) **Una definición de un puntero a un arreglo de 10 caracteres sin signo, llamado C.**
    unsigned char (*C)[10] = NULL;

30. **Describa con exactitud cada uno de los siguientes:**

    (a) **int (*I)[3];** I es un puntero a un array de 3 enteros con signo.

    (b) **static char* C[2];** C es un array de dos punteros a char con visibilidad local.

    (c) **extern char**C[2];** Se declara a C como un array de dos posiciones de punteros a punteros a char. C esta definido en un archivo externo.

31. **¿Qué es el polimorfismo? Ejemplifique mediante código**

Es una caracteristica de la programacion orientada a objetos en que todos los objetos que deciendan del mismo padre implementan al menos algun metodo en comun, pero agregandole un comportamiento caracteristico. Esto permite programar sin saber cual es el tipo exacto de objeto, pero si sabiendo su familia.

Ej:

```
class Animal {
   virtual int correr() = 0;
};

class Perro : public Animal {
   int correr() override {
       return 20; //En kh
   }
};

class Gato : public Animal {
   int correr() override {
       return 35; //En kh
   }
};

void imprimir_velocidad(Animal* animal) {
   std::cout << "Animal corriendo a: << animal->correr << "\n";
}

int main () {
   Perro perro;
   Gato gato;

   imprimir_velocidad(&perro);
   imprimir_velocidad(&gato);

   return 0;
}
```

32. **Escriba las siguientes definiciones/declaraciones:**

(a) **Declaración de un puntero a puntero a carácter**
char** a;

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

```
static long RESTA(unsigned long int a, unsigned long int b) {
    return a - b;
}
```

(c) **Definición de un entero corto sin signo solamente visible en el módulo donde se la define.**
static short int a = 1;

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

Deberia exigirle un archivo .h donde esten las declaraciones de las funciones y un arhivo (.cpp) donde se encuentren las definiciones de las funciones.

En la parte de linkeado se resuelven los porblemas de refernecia.

34. **Describa con exactitud las siguientes declaraciones/definiciones globales:**

    (a) **extern int (*I)[2];** Se declara a I como un puntero a un vector de dos posiciones de enteros. La definicion se encuentra en otro archivo.

    (b) **static char *C[3];** C es un arraz de punteros a char con visibilidad local.

    (c) **static float F(float a, float b);** F es la declaracion de una funcion que recibe dos float y devuelvo un float. Su visibilidad es local.

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

    Map es un contenedor del tipo clave-valor, siguiendo un orden especifico dado por la funcion de comparacion que se aplica a las claves. Las claves son unicas, es decir, existe una unica clave para cada valor.

    Ej:

```
std::map<char, int> mapa;
map.insert({'a', 1});          //Se agrega el valor 1 con clave a
std::cout << map['a'] << "\n"; //Se imprime 1
```

36. **Escriba las siguientes declaraciones/definiciones considerando un contexto global**

    (a) **La definición de un puntero a número flotante de doble precisión denominado F1**
    double* F1 = NULL;

    (b) **La declaración de un puntero a puntero a entero corto denominado F2**
    short int **F2;

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

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

    Una referencia en C++ es inmutable. Siempre apunta a un valor valido, nunca a null y no puede cambiarse al objeto que referencia. Se debe inicializadar al momento de crearse. Las principales diferencias con un puntero es que no se puede cambiar el valor adonde apunta, no puede referenciar a null, no se puede simplesmente declarar, se deben inicializar tambien.

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

    El template es una forma de crear codigo generico valido para cualquier tipo de dato. Permite reutilizar codigo, de manera que no se tenga que rededefinirlo para cada tipo de dato.

    Ej:

```
<template T>
void foo(T t) {
    std::cout << "Valor: " << t << "\n";
}

foo<int>(2); //Usamos la funcion con un entero de valor 2
foo(2);      //Igual que el anterior pero inferido.
```

39. **Escriba el .H correspondiente a una biblioteca que exporta:**

    (a) **La definición de un tipo llamado alumno_t correspondiente a una estructura con nombre (cadena de 50 caracteres) y padron (entero sin signo)**

    (b) **Una función llamada alumno_get_padron que toma un puntero a alumno_t y retorna un entero sin signo**

    (c) **Una función llamada procesar_alumnos que recibe un puntero a alumno_t, un entero sin signo y un puntero a función con parámetro a alumno_t y resultado vacío, 'procesar_alumnos' no debe retornar ningún valor**

```
#ifndef _ALUMNO_H_
#define _ALUMNO_H_

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

unsigned int alumno_get_padron(alumno_t* alumno);

void procesar_alumno(alumno_t* alumno, unsigned int a, void (*F)(alumnot));

#endif
```

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

    Una excepcion es un error que se da durante la ejecucion del programa. C++ nos da la posibilidad de lanzarlas y atraparlas como forma de manejar esos errores sin cortar la ejecucion (o si, caso sea necesario). Las clausulas try/catch cumplen exactamente ese proposito.

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

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

    En este caso, el string lanza una excepcion cuando tratamos de acceder a un valor que no exise (la posicion 100 no existe ya que solo posee 4 letras + "/0").

41. **¿Qué propósito tiene la función accept? Indique parámetros que recibe y retorno esperado**

    La funcion accept permite aceptar un nuevo cliente de la cola de espera. Nos devuelve un nuevo file descriptor que podemos usar para comunicarnos con el cliente.

    La firma de accept es: `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);`

    El retorno es un int que representa al file descriptor.

    - sockfd es un socket que se creó con socket(), bindeó a una dirección local con bind() y está esperando conexiones después de listen().

- addr: es un puntero a un struct sockaddr, donde se guardan las direcciones del peerskt.
- addrlen: en un principio contiene el tamaño en bytes de addr.

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

    Los programas escritos en C/C++ pueden ser compilados en distitas arquitecturas o sistemas operativos. Una aplicación escrita en C es portable a nivel de código fuente, que luego podra ser compilado en el otro entorno y asi generar un ejecutable para ese entorno.

    La precaución a tomar debe ser tener instaladas las bibliotecas de desarrollo utilizadas en el sistema operativo en el que se compilará.

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

    Un hilo contiene un PC, un stack y los registros del procesador por separado, pero comparte los file descriptors, el código del programa (code segment) y todos los demás segmentos (data, heap) a excepción del stack, como se mencionó. En cambio el proceso tiene todo esto por separado. La conveniencia de usar uno u otro es relativo y depende mucho del nivel de separación que se requiera, por ejemplo si tenemos un mismo programa que vaya a realizar distintas tareas concurrentemente pero opere sobre los mismos datos o incluso ejecute el mismo código de forma paralela o concurrente, probablemente necesitemos hilos. En cambio si necesitamos realizar diferentes tareas mucho más separadas que eventualmente podrían requerir comunicarse, pero que nunca van a acceder a los mismos datos ni van a ejecutar el mismo código, podríamos utilizar procesos separados.

44. **Explique cómo funciona la sobrecarga de operadores en C++** La sobrecarga de operadores consiste en definir un operador a un tipo de dato/objeto que hayamos creado. Esto nos permite trabajar con mas naturalidad ademas de un codigo mas legible. Por ejemplo:

```
typedef struct Complex {
    int re;
    int im;

    Complex& operator+(const Complex& other) {
       this->re += other.re;
       this->im += other.im;
       return *this;
    }
}
```

45. **Explique qué son los métodos const en C++ y para qué sirven. De un breve ejemplo donde su uso sea imprescindible.** Los metodos const aseguran que no modifican el estado interno de un objeto. Esto es muy util ya que permiten detectar errores/bugs en tiempo de compilacion, ya que se si trata de modifcar un objeto dentro de un metodo const, el error salta al compilar.

    Un caso imoprecindible es en la sobrecarga del operador ==. Esta sobrecarga no debe modificar al objeto.

```
bool operator==(const Objeto& other) const;
```

46. **Describa y ejemplifique el uso de la siguiente instrucción de precompilación: #define**

    Define define un simbolo al compilador. Esto es, en el proceso de precompilacion, cada vez que el compilador encuentre el simbolo del define, lo reemplazara por su valor. AL simbolo del define se le puede asignar tanto valores literales como un bloque de codigo (macros).

    Ejemplo de define literal.

```
#define PI 3.14

void foo() {
    std::cout << PI << "\n";
}
```

Ejemplo de define con codigo.

```
#define SQUARE(X) (X)*(X)

void foo() {
    std::cout << SQUARE(2) << "\n";
}
```

47. **¿Qué es un iterador de la librería estándar de C++? Ejemplifique su uso** Un iterador es un objeto que permite recorrer una coleccion, independizandonos del contenedor. Es muy util ya que es psoible escribir un codigo independiente al tipo de contenedor que se utilize.

    Ej:

```
std::vector<int> vector;

for (auto i = vector.begin(); i != vector.end(); i++)
    std::cout << *i << "\n";
```

    Si cambiáramos std::vector por std::list, el código seguiría corriendo igual.

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

    En la programacion orientada a enventos estamos cosntantemente a la espera de un nuevo evento. Como no sabemos que evento va a suceder, se debe primero decodificar el evento y luego manejarlo.

    Pseudocodigo loop eventos:

```
seguir_recibiendo = true;
mientras seguir_recibiendo

    evento = recibir_evento.
    evento.decodificar();

    if evento == salir
        seguir_recibiendo = false
    else
        evento.manejar();
```

    En la mayoria de los casos (y en los lenguages que lo permiten), el manejo de cada evento se maneja de forma polimorfica, donde cada evento sabe como aplicarse al programa.

49. **Explique cómo funciona la herencia pública en C++ y en qué se diferencia de la herencia privada**

    La herencia publica permite a los hijos de la clase acceder a los metodos y atributos publicos y protegidos de la clase padre. Si la herencia es privada, todos los metodos y atirbutos pasan a ser privados en la clase hija, es decir, no podrán se accedidos por medio de ella.

50. **¿Qué es un functor? Ejemplifique**

    Un functor es un objeto que posee sobrecargado el operador llamado (). Este tipo de objetos desacoplar el momento en que se le pasa los parámetros a una función del momento en que se inicia la ejecución de la misma.

Un uso muy comun es para pasarle una tarea a un thread.

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

class Sumar {
    int a;
    int b;
    int resultado;

    Sumar(int a, int b) a(a), b(b) {}

    void operator()() {
        resultado = a + b;
    }

    int obtener_resultado() {
        return this->resultado;
    }
}

int main() {
    Sumar suma(1,2);

    std::thread sumador(std::ref(suma));
    sumador.join();

    std::cout << suma.objetener_resultado() << "\;

}
```

51. **¿Qué es un parámetro opcional/default en C++? ¿Cómo se utiliza? ¿Dónde puede usarse? Ejemplifique**

    Un parametro ocpional es un parametro que posee un valor por defector. Si el parametro se envia, este toma el nuevo valor, caso contrario utiliza el valor por defecto. Los parametros opcionales tienen que estar siempre al final. Luego de un parametro opcional, todos los demas seran opcionales.

    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

```
//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
```

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

El objetc slicing es cuando un objeto de la clase derivada se copia a un objeto de la clase base, perdiendo sus atributos de clase derivada.

Ejemplo:

```
class Base {

    int a;

    Base(int a) a(a) {};

    void print() {
        std::cout << "A " << a << std::endl;
    }

};

class Derivada : public Base {

    int b;

    Derivada(int a, int b) Base(a), b(b) {};

    void print() {
        std::cout << "A " << a << std::endl;
        std::cout << "B " << b << std::endl;
    }
}

int main() {
    Derivada derivada(3,4);
    Base base = derivada;
    //Aca ya sucedio el slicing

    base.print();
    //salida-> "A 3";

    return 0;
}
```

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

Un 'proceso' es una entidad de ejecución independiente, donde, el sistema operativo, en el momento en que el proceso se lanza, proporciona un espacio de direcciones de memoria en los que el proceso puede ejecutarse.

Los hilos son entidades de ejecución independiente que viven dentro de los procesos y, por tanto, viven dentro del mismo espacio de direcciones de memoria que otros hilos, lo que permite acceder a cualquier dato dentro del mismo proceso.

Los procesos comparten solamente el DataSegment y el CodeSegment. Los registros, stack y heap no son compartidos.

Por su vez, los hilos comparten ademas del DataSegment y CodeSegment, comparten el heap.

Cuando se desea trabajar con paralelismo compartiendo memoria es muccho mas facil y barato el uso de hilos. El proceso de context switching entre hilos y del compartido de memoria es mucho mas facil de manejar para el SO que entre procesos. Los procesos se deben usar cuando no se desea compartir el heap. En ese caso, son mucho mas seguros los procesos.

54. **Explique en qué situaciones es recomendable utilizar programación multi-hilo(multithreading) para realizar cierto procesamiento. ¿Existe algún caso donde utilizar multithreading sea perjudicial?**

El multithreading puede ser muy util a la hora de querer mejorar la performace de un programa, pero no siempre el uso de threads implica mayor velocidad. Los hilos se deben usar cuando tenemos opreaciones bloqueantes y decidimos seguir con la ejecucion o cuando tenemos algunas operaciones que demoran mas de lo normal y por lo tanto podemos aprovechar ese tiempo para realizar otros porcesamientos.

El uso de threads puede ser perjudicial en varios casos. El mas comun es cuando se queire forzar el uso en situaciones que no son necesarios, haciendo un codigo mucho mas suceptile a errores/bugs.

Es necesario ener el cuenta el costo del context switch vs el costo de la tarea. Si el costo del context switch es mayor que el de la tarea, no conviene la utilizacion de threads.

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

Una función callback es una función pasada dentro de otra función como parámetro y que luego es invocada dentro de esa función para completar alguna rutina o acción. Es importante en entornos gráficos porque se puede ejecutar en respuesta a una acción predeterminada. Cada accion puede tener un callback asociado. Esto permite un codigo mas generico.

```
void for\_each(std::list<int>& list, void (F*) (int&)) {
    for (auto& elem : list)
        F(elem);
}
```

56. **Escriba y ejemplifique el uso de la siguiente instrucción de precompilación: #ifdef**

Ifdef significa “si lo que sigue está definido” mientras que ifndef significa “si lo que sigue no está definido”.

Ej:

```
#define PI 3.1416
#ifdef PI
printf("%i\n", PI);
#else
printf("3.1416\n");
#endif
```

Al codigo anterior entra a la primer condicion ya que PI esta definido.

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

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

Por lo que crea el símbolo en caso de que no esté definido. De esta manera, en una primera corrida solo se crea el símbolo la primera vez y el resto de las veces no.

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

Durante el proceso de compilacion, el preprocesador remplaza el #include por el contenido del archivo que sigue. De esa manera se obtienen las delcaraciones de las funciones a utilizar.

No se recomienda el #include de archivos .c ya que no poseen los protectores que los headers tienen. Al declarar un header, se usa la directiva:

```
#ifndef _LIRERIAL_H_
#define _LIRERIAL_H_
```

Esto evita que se incluya mas de una vez las declaraciones en el mismo archivo, ya que si estan definidas no las incluye. Los archivos .c no tienen esta proteccion, podiendo darse una doble de los metodos/funciones.

2 Notas

- 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 descendientes.
- Los atributos/métodos privados sólo pueden ser accedidos en la clase donde se definen