Conociendo los archivos.h

Antes de trabajar con archivos header en C++, es importante tener un conocimiento sólido de varios temas fundamentales del lenguaje, ya que los archivos header son una parte más avanzada del proceso de organización del código. Aquí te dejo los temas clave que debes dominar antes de trabajar con archivos header:

1. Fundamentos de Programación en C++
Sintaxis básica: Conocer cómo se estructuran los programas en C++ y la forma correcta de escribir código.
Tipos de datos y variables: Saber cómo declarar y usar tipos de datos primitivos como int, float, double, char, bool.
Operadores: Conocimiento de operadores aritméticos, lógicos, relacionales, de asignación, etc.
Estructuras de control: if, else, switch, for, while, do-while, y cómo controlar el flujo del programa.

2. Funciones
Declaración e implementación: Entender cómo se declaran e implementan funciones en C++, lo que es esencial porque los archivos header normalmente contienen declaraciones de funciones, mientras que la implementación se encuentra en los archivos fuente.
Prototipos de funciones: Los archivos header solo contienen los *prototipos de las funciones*, así que debes comprender cómo funciona esta técnica (declarar una función antes de implementarla).
Paso de parámetros: Conocer las diferencias entre pasar parámetros por valor y por referencia en las funciones.
Funciones inline: Saber cómo y cuándo usar funciones inline, ya que a veces se colocan en los archivos header para mejorar el rendimiento.

3. Concepto de Compilación y Enlaces
Compilación y Linker: Comprender el proceso de compilación en varias fases. Los archivos header están involucrados en la etapa de preprocesamiento y compilación.
Inclusión de archivos: Conocer el uso de #include, ya que los archivos header se incluyen mediante esta directiva.
Directivas del preprocesador: Entender directivas como #define, #ifndef, #ifdef, y #endif, que son fundamentales para evitar la inclusión múltiple de archivos header.
Enlace externo: Saber cómo funciona el linkage de las funciones y variables en diferentes archivos del proyecto.

4. Estructuras y Clases
Estructuras (struct): Comprender cómo funcionan las estructuras en C++, ya que suelen declararse en archivos header.
Clases y Programación Orientada a Objetos (POO): Tener un buen conocimiento sobre la Programación Orientada a Objetos (POO), ya que los archivos header se usan frecuentemente para declarar clases.
Encapsulamiento: Declarar y proteger datos privados en las clases.
Constructores y destructores: Conocer cómo se declaran e implementan en archivos separados.
Herencia: Saber cómo usar herencia para definir relaciones entre clases en diferentes archivos.
Polimorfismo: Especialmente si se trabaja con funciones virtuales en archivos header.

5. Espacios de nombres (namespace)
Uso de namespace: Los archivos header se benefician mucho del uso de espacios de nombres para evitar colisiones de nombres en proyectos grandes. Es importante entender cómo definir y usar un namespace.

6. Constantes, Macros y Definiciones
Constantes (const): Saber cómo definir constantes globales o relacionadas con clases que a menudo se declaran en archivos header.
Macros: Comprender cómo funcionan las macros con #define, aunque se deben usar con cuidado.
Enumeraciones (enum): El uso de enumeraciones puede ser importante al trabajar con archivos header para definir conjuntos de valores constantes.

7. Declaraciones y Definiciones
Declaración vs Definición: Comprender la diferencia entre una declaración (lo que aparece en los archivos header) y una definición (que aparece en los archivos fuente).
Declaración: Solo informa al compilador que una función o clase existe.
Definición: Proporciona el cuerpo de la función o el código de la clase.

8. Bibliotecas estándar de C++ (STL)
Contenedores (vectores, listas, etc.): Debes entender cómo funcionan los contenedores como std::vector, std::list, std::map, ya que pueden declararse y usarse en archivos header.
Algoritmos y utilidades: Conocer los algoritmos estándar (std::sort, std::find) y las utilidades (std::pair, std::tuple).

9. Manejo de Punteros

Punteros: Saber manejar punteros correctamente, ya que es muy común trabajar con punteros en archivos header, especialmente al declarar funciones o clases que manipulan memoria dinámicamente.
Memoria dinámica: Conocer el uso de los operadores new y delete y cómo gestionar la memoria dinámica.

10. Plantillas (Templates)
Funciones y clases plantilla: Los archivos header se usan frecuentemente para declarar y definir templates de funciones y clases, ya que estos deben definirse completamente en el archivo header para que el compilador pueda instanciar el código para cada tipo que se utilice.

11. Compilación y Ejecución de Programas Multi-archivo
Compilación de programas grandes: Es importante comprender cómo compilar proyectos que tienen múltiples archivos fuente (.cpp) y archivos header (.h), ya que en proyectos grandes es necesario compilar varios archivos separados y luego enlazarlos.

Los archivos header (o archivos de cabecera) en C++ son archivos que contienen declaraciones de funciones, clases, variables, constantes y otras entidades que luego pueden ser utilizadas en diferentes archivos del programa. Los archivos header suelen tener la extensión .h, y son esenciales para organizar y modularizar el código.

¿Por qué utilizar archivos header?
Reutilización de código: Puedes declarar funciones, clases o variables en un archivo .h y luego usarlas en varios archivos fuente (.cpp).
Modularización: Los archivos header permiten dividir un programa grande en partes más pequeñas y manejables.
Separación de declaración e implementación: La declaración de funciones y clases se pone en los archivos header, mientras que la implementación real (el código) se coloca en archivos fuente (.cpp).

Estructura básica de un archivo header
1. Declaraciones de funciones.
2. Declaraciones de clases.
3. Definiciones de constantes y macros.
4. Sintaxis básica de un archivo header

Un archivo header solo contiene declaraciones, no definiciones completas. Por ejemplo:

```
// archivo "milib.h"
#ifndef MILIB_H  // Verificación para evitar la inclusión múltiple
#define MILIB_H

// Declaración de una función
void saludar();

// Declaración de una clase
class Persona {
public:
     void mostrarNombre();
};

#endif   // Fin de la verificación de inclusión múltiple
```

Este archivo no contiene la implementación de la función saludar o la función mostrarNombre de la clase Persona. Solo las declara.

Explicación de la verificación de inclusión múltiple
El preprocesador usa directivas para asegurarse de que un archivo header no sea incluido varias veces en un mismo archivo fuente, lo que causaría errores de compilación. Esto se hace usando:

#ifndef: "Si no está definido..."
#define: Define una etiqueta única.
#endif: Finaliza el bloque de preprocesador.

Implementación en el archivo fuente (.cpp)
El archivo fuente correspondiente contiene la implementación de las funciones declaradas en el header:

```
// archivo "milib.cpp"
#include <iostream>
#include "milib.h"

// Implementación de la función saludar
void saludar() {
     std::cout << "¡Hola desde la función saludar!" << std::endl;
```

```
}

// Implementación de la función de la clase Persona
void Persona::mostrarNombre() {
     std::cout << "¡Soy una persona!" << std::endl;
}
```

Uso en el archivo principal
En el archivo principal de tu programa, puedes incluir el archivo header e invocar las funciones que has declarado en él:

```
// archivo "main.cpp"
#include <iostream>
#include "milib.h"

int main() {
     // Llamar a la función saludar
     saludar();

     // Crear un objeto de la clase Persona
     Persona persona;
     persona.mostrarNombre();

     return 0;
}
```

Compilación del programa
Cuando compilas este tipo de programas con archivos separados, debes compilar todos los archivos .cpp. Si estás usando un compilador en la línea de comandos como g++, puedes hacer lo siguiente:

```
g++ main.cpp milib.cpp -o programa
```

Esto creará un ejecutable llamado programa que puedes ejecutar:

```
./programa
```

Ejemplo completo

Archivo header (milib.h):

```
#ifndef MILIB_H
#define MILIB_H

void saludar();

class Persona {
public:
     void mostrarNombre();
};

#endif
```

Archivo fuente (milib.cpp):

```
#include <iostream>
#include "milib.h"

void saludar() {
     std::cout << "¡Hola desde la función saludar!" << std::endl;
}

void Persona::mostrarNombre() {
     std::cout << "¡Soy una persona!" << std::endl;
}
```

Archivo principal (main.cpp):

```
#include <iostream>
#include "milib.h"

int main() {
     saludar();

     Persona persona;
```

```
    persona.mostrarNombre();

    return 0;
}
```

Ventajas del uso de archivos header
Modularidad: Puedes organizar el código en múltiples archivos, facilitando su mantenimiento.
Reutilización: El mismo archivo header puede ser utilizado en varios proyectos.
Legibilidad: Al separar las declaraciones de las implementaciones, el código se vuelve más fácil de entender.

Buenas prácticas

Verificación de inclusión múltiple: Siempre usa las directivas #ifndef, #define y #endif para evitar problemas de inclusión múltiple.
Solo declarar, no definir: Un archivo header debe contener solo declaraciones, no definiciones. Las definiciones deben estar en los archivos .cpp.
Documentación: Agrega comentarios a las funciones y clases declaradas en los archivos header *para que otros desarrolladores sepan cómo usarlas* sin necesidad de revisar las implementaciones.

Resumen
Los archivos header .h son utilizados para declarar funciones, clases y variables que se definen en otros archivos.
Ayudan a dividir el código en módulos, lo que mejora la organización y facilita la colaboración en proyectos grandes.
Al incluir un archivo header en varios archivos fuente, puedes reutilizar código de manera eficiente.