Header Files in C

What are header files in C?

Header files in C are files that generally include declarations for functions, macros, constants, and data types, which can be used across multiple source files. They enable the separation of the interface (declarations) from the implementation (definitions) and help enhance code reusability, maintainability, and modularity. Header files commonly have the .h extension.

Syntax of Header Files in C

We can include header files in C by using one of the given two syntax whether it is a predefined or user-defined header file.

```
#include <filename.h>    // for files in system/default directory
      or
#include "filename.h"    // for files in same directory as source file
```

Example of header file in c:

```
#include <stdio.h>
int main()
{
   printf(
       "Printf() is the function in stdio.h header file");
   return 0;
}
```

Types of C Header Files

There are two types of header files in C:

- Standard / Pre-existing header files
- Non-standard / User-defined header files

Standard/Pre-existing header files

Standard header files in C are predefined files provided by the C Standard Library, which contain declarations for various functions, macros, and data types that facilitate common programming tasks. Here’s a list of some standard header files and their uses:

1. <stdio.h>

Use: Contains functions for input and output operations.
Common Functions:
printf(): Used for outputting formatted text to the console.
scanf(): Used for reading formatted input from the console.
fopen(), fclose(): Used for file operations.

2. <stdlib.h>
Use: Provides functions for memory allocation, process control, conversions, and other utility functions.
Common Functions:
malloc(), calloc(), free(): Used for dynamic memory management.
exit(): Used to terminate a program.
atoi(), atof(): Used for converting strings to integers and floating-point numbers, respectively.

3. <string.h>
Use: Contains functions for manipulating strings.
Common Functions:
strcpy(): Copies a string.
strlen(): Returns the length of a string.
strcat(): Concatenates two strings.

4. <math.h>
Use: Provides mathematical functions.
Common Functions:
sin(), cos(), tan(): Trigonometric functions.
sqrt(): Calculates the square root.
pow(): Raises a number to a power.

5. <time.h>
Use: Contains functions for manipulating and formatting date and time.
Common Functions:
time(): Returns the current time.
strftime(): Formats time and date.
clock(): Returns processor time used by the program.

6. <ctype.h>
Use: Provides functions for character classification and conversion.
Common Functions:
isalpha(): Checks if a character is an alphabetic letter.
isdigit(): Checks if a character is a digit.
toupper(), tolower(): Converts characters to uppercase or lowercase.

7. <limits.h>
Use: Defines macros that specify the implementation-defined limits of various data types.
Common Macros:
INT_MAX, INT_MIN: Limits for the int type.
CHAR_BIT: Number of bits in a byte.

8. <float.h>
Use: Defines macros for floating-point type limits.
Common Macros:
FLT_MAX, FLT_MIN: Limits for the float type.
DBL_MAX, DBL_MIN: Limits for the double type.

Preprocessors in C

In C programming, preprocessors are directives that give instructions to the compiler to preprocess the information before actual compilation starts. They start with a # symbol and are handled by the preprocessor, which operates on the code before it is compiled. Preprocessing directives are typically used to define constants, include files, and conditionally compile code.

Some common preprocessor directives

Here are some of the common preprocessor directives in C:

Preprocessor Directives	Description
#include	Used to include a file in the source code program
#define	Used to define a macro
#undef	Used to undefine a macro
#ifdef	Used to include a section of code if a certain macro is defined by #define
#ifndef	Used to include a section of code if a certain macro is not defined by #define
#if	Check for the specified condition
#else	Alternate code that executes when #if fails
#elif	Combines else and if for another condition check
#endif	Used to mark the end of #if, #ifdef, and

	#ifndef

Types of C Preprocessors

There are 4 Main Types of Preprocessor Directives:

- Macros
- File Inclusion
- Conditional Compilation
- Other directives

1. Macro Preprocessors
Macro preprocessors define macros, which are pieces of code that can be reused throughout the program. A macro is typically defined using the #define directive. Macros can either be simple constants or more complex functions.
Constant Macros:
#define PI 3.14159
Function-like Macros:
#define SQUARE(x) (x * x)

2. File Inclusion Preprocessors
This type of preprocessor is used to include the contents of one file into another file. The #include directive is used for this purpose. It is mainly used to include header files that contain function prototypes, constants, and macros.
Syntax:
#include <stdio.h> // Include a standard library header
#include "myheader.h" // Include a user-defined header

3. Conditional Compilation Preprocessors
Conditional compilation preprocessors allow certain parts of the code to be compiled or ignored depending on specific conditions. This is useful for including or excluding code based on environment, platform, or debugging requirements.
Directives used:
#ifdef – Compiles the code if a certain macro is defined.
#ifndef – Compiles the code if a certain macro is not defined.
#if / #elif / #else – Allows for more complex conditional checks.
#endif – Ends a conditional block.

4. Other Special Preprocessors
These preprocessors perform special tasks that are not directly related to macros, file inclusion, or conditional compilation. They include:
#pragma: Provides compiler-specific instructions. It is often used to control optimization, suppress warnings, or alter the behavior of the compiler for specific sections of code.

```
#pragma warning(disable : 4996)  // Disable a specific warning in Visual Studio
```

#undef: This directive is used to undefine a macro that was previously defined, making it invalid for further use.
Syntax:#undef PI
#error: Generates an error message during compilation if a specific condition occurs.
Syntax: #error "Compilation failed due to incompatible version"
#line: Used to change the line numbers reported by the compiler for error or warning messages.
Syntax: #line 100 "newfile.c"