**ASSIGNMENT**

**2). Compiler –** is a software tool that translates human-readable source code written in a high-level programming language, such as C, into machine-readable code (object code) that a computer can execute.

**Source code-** refer to the original human-readable program written in a high-level programming language such as C.

**Object code-** is a low-level machine-readable representation of a program generated by the compiler during the compilation process.

**Linkers-** are tools or components of development environment responsible for combining multiple object code files and libraries into a single executable program.

**3).** Example:

#include <stdio.h>

int main () {

int num1 = 6;

int num2 = 8;

int sum = num1 + num2

printf("The sum is: %d\n", sum);

return 0;

}

**Compilation process**

**1. Writing the source code:** You write the C program in a text editor or integrated development environment (IDE). In this case, you’ve created a C program that adds two numbers and prints the result.

**2. Pre-processing:** Before the actual compilation, the source code goes through a pre-processing stage. The pre-processor, which is part of the compiler, handles directives starting with **#**, such as **#include**. I t process header files and macros and prepares the code for compilation. In your code, the **#include**. It processes header files and macros and prepares the code for compilation. In your code, the **#include<stdio.h>** directive is processed, including the contents of the standard I/O library (stdio.h) in your program.

**3. Compilation:** The C compiler (e.g., GCC) translates the pre-processed source code into intermediate object code. During this phase, the compiler performs lexical analysis, syntax analysis, and generates object code. The object code produced is specific to the target system’s architecture. In this case, compiler generates machine code that performs the addiction and includes code for the “printf” function from the standard library.

**4. Linking:** After compilation, you have object code, but it might still contain references to functions or libraries that are not part of your source code. The linker, which is part of the compiler toolchain, is responsible for resolving these references. It links your object code with the necessary standard libraries and other object code files if you're using external libraries. In this simple program, it would link with the standard C library, which contains the implementation of **printf.** The linker combines all the necessary object files into an executable file.

**5. Executable output:** Once the linker completes its work, you get an executable file. In Unix-like systems, this file typically has no file extension, and you can run it from the command line. In Windows, it has a **.exe** extension. In this case, you'd have an executable file that yse adds two numbers and prints the result when run.

4. **Differences between a compiler and an interpreter.**

|  |  |
| --- | --- |
| **Compiler** | **Interpreter** |
| 1.Converts the entire program into machine code as a whole | 1.Converts a program into machine code statement by statement |
| 2.It takes large amount of time to analyse the source code but the so the overall execution time is faster | 2.Takes less amount of time to analyse the source code but the overall execution time is slower |
| 3.it generates intermediate source code | 3.No intermediate object code is generated |
| 4.Memory requirement is more since object code is generated | 4.Memory requirement is less |
| 5.Error detection is difficult | 5.Error detection is easy |
| 6.Examples; C, C++, Go, Rust | 6.Examples; Python, Perl, Ruby, PHP, JavaScript |

**5. Arithmetic Operators:**

**+**(Addition)

**-**(Subtraction)

\*(multiplication)

/(Division)

% (Modulus, remainder of division)

++(Increment)

--(Decrement)

2. **Relational Operators:**

== (Equal to)

!= (Not equal to)

< (Less than)

> (Greater than)

<= (Less than or equal to)

>= (Greater than or equal to)

**3. Logical Operators:**

&& (Logical AND)

|| (Logical OR)

! (Logical NOT)

**4.Bitwise Operators:**

& (Bitwise AND)

| (Bitwise OR)

^ (Bitwise XOR)

~ (Bitwise NOT, one’s complement)

<< (Left shift)

>> (Right shift)

**5. Assignment Operator:**

= (Assignment)

+= (Addition assignment)

-= (Subtraction assignment)

\*= (Multiplication assignment)

/= (Division assignment)

%= (Modulus assignment)

&= (Bitwise AND assignment)

|= (Bitwise OR assignment)

^= (Bitwise XOR assignment)

<<= (Left shift assignment)

>>= (Right shift assignment)

**6. Conditional (Ternary) Operator:**

?: (Conditional operator)

**7. Miscellaneous Operators:**

sizeof (Size of a data type or object)

& (Address-of operator)

\* (Pointer dereference operator)

, (Comma operator)

-> (Structure pointer operator)

. (Structure member access operator)

**8. Unary Operators:**

+ (Unary plus)

- (Unary minus)

++ (Increment)

-- (Decrement)

! (Logical NOT)

~ (Bitwise NOT)

& (Address-of)

\* (Pointer dereference)

**9. Relational Operators:**

sizeof (Size of a data type or object)

& (Address-of)

\* (Pointer dereference)

[] (Array subscript)

() (Function call)