Judy natasha Wambui gachanja.

Sct222-0134/2023

Introduction to programming assignment.

1. Define the following terms as used in C PROGRAMMING.

* **Compiler** – It is a special program that translates a programming language’s source code into machine code, bytecode or another programming language.
* **Source code** – Programming statements that are created by a programmer with a text editor or a visual programming tool and then saved in a file.
* **Object code** – The output, a compiled file, which is produced when the source code is compiled by a compiler.
* **Linkers** – They are important utility programs that take the object files, produced by compilers and assemblers, and other codes to join them into single executable files.

2.Using an example, I.E A PROGRAM TO ADD TWO NUMBERS, EXPLAIN THE COMPILATION PROCESS OF A C PROGRAM.

The compilation process in C involves four steps:

* Pre-processing
* Compiling
* Assembling
* Linking

Code for adding two numbers:

#include <stdio.h>

int main() {

int number1, number2, sum;

printf("Enter first number: ");

scanf("%d", &number1);

printf("Enter second number: ");

scanf("%d", &number2);

sum = number1 + number2;

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

return 0;

}

Pre-processing

The preprocessor is responsible for handling directives starting with #, such as #include. It includes the contents of the specified header file (stdio.h in this case) into the source code.

The resulting code after preprocessing is called the "preprocessed code."

Compiling

The compiler translates the preprocessed code into assembly code. This is a low-level representation of the program, specific to the target architecture.

Assembling

The assembler then converts the assembly code into machine code (binary code) specific to the computer's architecture. This step produces an object file (with a .o or .obj extension).

Linking

The linker takes care of combining the object file with other necessary object files (like standard libraries) to create an executable file.

In our example, the linker would include the implementation of the functions from the standard input/output library (printf and scanf) and other necessary system-level code.

3.EXPLAIN THE DIFFERENCES BETWEEN A COMPILER AND AN INTERPRETER.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Compiler** | **Interpreter** |
| Steps of Programming | * Creation of the program. * The Compiler analyses all the language statements and throws an error when it finds something incorrect. * If there’s zero error, the compiler converts the source code to machine one. * It links various code files into a runnable program (exe). * It runs the program. | * Creation of the program. * It doesn’t require the linking of files or generation of machine code. * It executes the source statements line by line during the execution. |
| Advantage | The code execution time is comparatively less because the program code already gets translated into machine code. | They are fairly easy to use and execute, even for a beginner. |
| Disadvantage | One can’t change a program without getting back to the source code. | Only computers with the corresponding Interpreter can run the interpreted programs. |
| Machine Code | It stores the machine language on the disk in the form of machine code. | It doesn’t save the machine language at all. |
| Running Time | The compiled codes run comparatively faster. | The interpreted codes run comparatively slower. |
| Model | It works on the basis of the language-translation linking-loading model. | It works on the basis of the Interpretation method. |
| Generation of Program | It generates an output program in the exe format. A user can run it independently from the originally intended program. | It doesn’t generate an output program. Meaning, it evaluates the source program every time during individual execution. |
| Execution | One can separate the program execution from the compilation. Thus, you can perform it only after completing the compilation of the entire output. | Execution of the program is one of the steps of the Interpretation process. So, you can perform it line by line. |
| Memory Requirement | Target programs execute independently. They don’t require the Compiler in the memory. | Interpreter originally exists in the memory at the time of interpretation. |
| Best Fitted For | You cannot port the Compiler because it stays bound to the specific target machine. The compilation model is very common in programming languages like C and C++. | They work the best in web environments- where the load time is very crucial. Compiling takes a relatively long time, even with small codes that may not run multiple times due to the exhaustive analysis. Interpretations are better in such cases. |
| Optimization of Code | A compiler is capable of seeing the entire code upfront. Thus, it makes the codes run faster by performing plenty of optimizations. | An interpreter sees a code line by line. The optimization is, thus, not very robust when compared to Compilers. |
| Dynamic Typing | Compilers are very difficult to implement because they can’t predict anything that happens during the turn time. | The Interpreted language supports Dynamic Typing. |
| Use | It works best for the Production Environment. | It works the best for the programming and development environment. |
| Execution of Error | A Compiler displays every error and warning while compiling. So, you can’t run this program unless you fix the errors. | An Interpreter reads every statement, then displays the errors, if any. A user must resolve these errors in order to interpret the next line. |
| Input | A Compiler takes a program as a whole. | An Interpreter takes single lines of a code. |
| Output | The Compilers generate intermediate machine codes. | The Interpreters never generate any intermediate machine codes. |
| Errors | This translator displays all the errors after compiling- together at the same time. | It displays the errors of every single line one by one. |
| Programming Languages | Java, Scala, C#, C, C++ use Compilers. | Perl, Ruby, PHP use Interpreters. |

List all the main categories of operators available in C PROGRAMMING.

C divides the operators into the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Bitwise operators

Arithmetic Operators

* Arithmetic operators are used to perform common mathematical operations.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** |
| + | Addition | Adds together two values | x + y |
| - | Subtraction | Subtracts one value from another | x - y |
| \* | Multiplication | Multiplies two values | x \* y |
| / | Division | Divides one value by another | x / y |
| % | Modulus | Returns the division remainder | x % y |
| ++ | Increment | Increases the value of a variable by 1 | ++x |
| -- | Decrement | Decreases the value of a variable by 1 | --x |

## Assignment Operators

Assignment operators are used to assign values to variables.

A list of all assignment operators:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| \*= | x \*= 3 | x = x \* 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| &= | x &= 3 | x = x & 3 |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

## Comparison Operators

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

The return value of a comparison is either 1 or 0, which means **true** (1) or **false** (0). These values are known as **Boolean values.**

A list of all comparison operators:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| == | Equal to | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

Logical Operators

You can also test for true or false values with logical operators.

Logical operators are used to determine the logic between variables or values:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** |
| && | Logical and | Returns true if both statements are true | x < 5 &&  x < 10 |
| || | Logical or | Returns true if one of the statements is true | x < 5 || x < 4 |
| ! | Logical not | Reverse the result, returns false if the result is true | !(x < 5 && x < 10) |

Bitwise operators

The Bitwise operators are used to perform bit-level operations. The operators are first converted to bit-level and then the calculation is performed. Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing.

|  |  |
| --- | --- |
| **Symbol** | **Operator** |
| & | bitwise AND |
| | | bitwise inclusive OR |
| ^ | bitwise XOR (exclusive OR) |
| << | left shift |
| >> | right shift |
| ~ | bitwise NOT ([ones' complement](https://en.wikipedia.org/wiki/Ones%27_complement)) (unary) |