2016

(1) (a) Diagram (x)

3 (a) Slide

4 (c) Slide

5 (a) Slide - 2017 - 5 (a)

(c) Slide

6 (a) - 2019 - 1 (B)

Q 7 (a)

① sort()

→ Syntax,

sort (First, last)

ex- int myints [ ] = $\{\underbrace{5,7,9,1,4,3}_{6}\}$;

vector <int> myvct (myints, myints + 6);

sort (myvct. begin(), myvct. end())

Purpose
sorts the elements in the range (first, last) into ascending order.

② strcmp()
syntax -

~~strcmp(left hand~~
strcmp(str1 , str 2);

~~return~~ [illegible]

return - 0 - equal string
- positive - str1 > str 2
- negative str1 < str 2

Perpose.-
used for string comparison.

③ ~~[illegible]~~ strcpy ()

Syntax -

~~strcpy(from,~~

strcpy (destination , from)

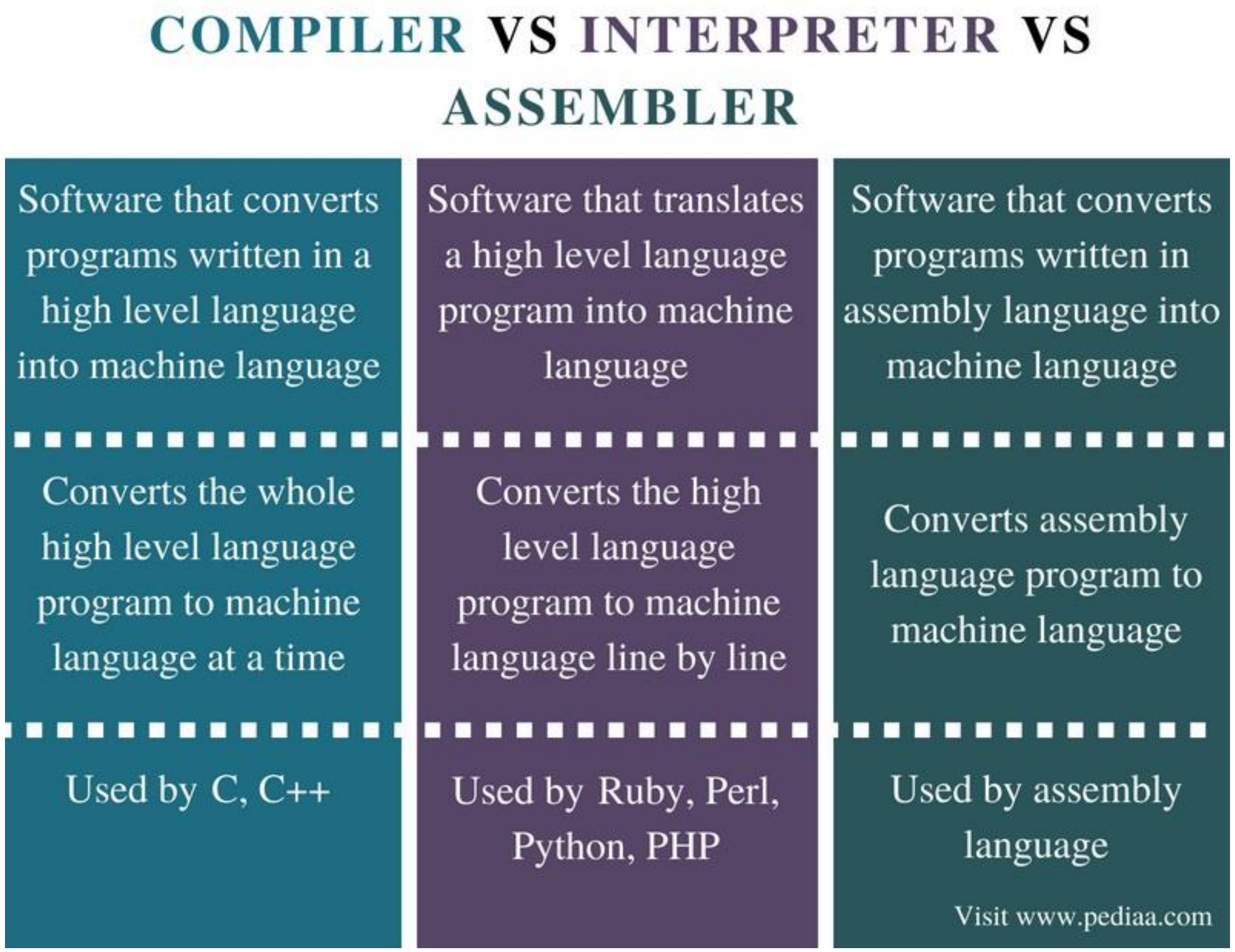
returns → the string saved into 'from' to the 'destination' string.

perpose -

string copying, from one variable to another.

2016 Q/A Slon

3 c



COMPILER VS INTERPRETER VS ASSEMBLER

Software that converts programs written in a high level language into machine language	Software that translates a high level language program into machine language	Software that converts programs written in assembly language into machine language
Converts the whole high level language program to machine language at a time	Converts the high level language program to machine language line by line	Converts assembly language program to machine language
Used by C, C++	Used by Ruby, Perl, Python, PHP	Used by assembly language

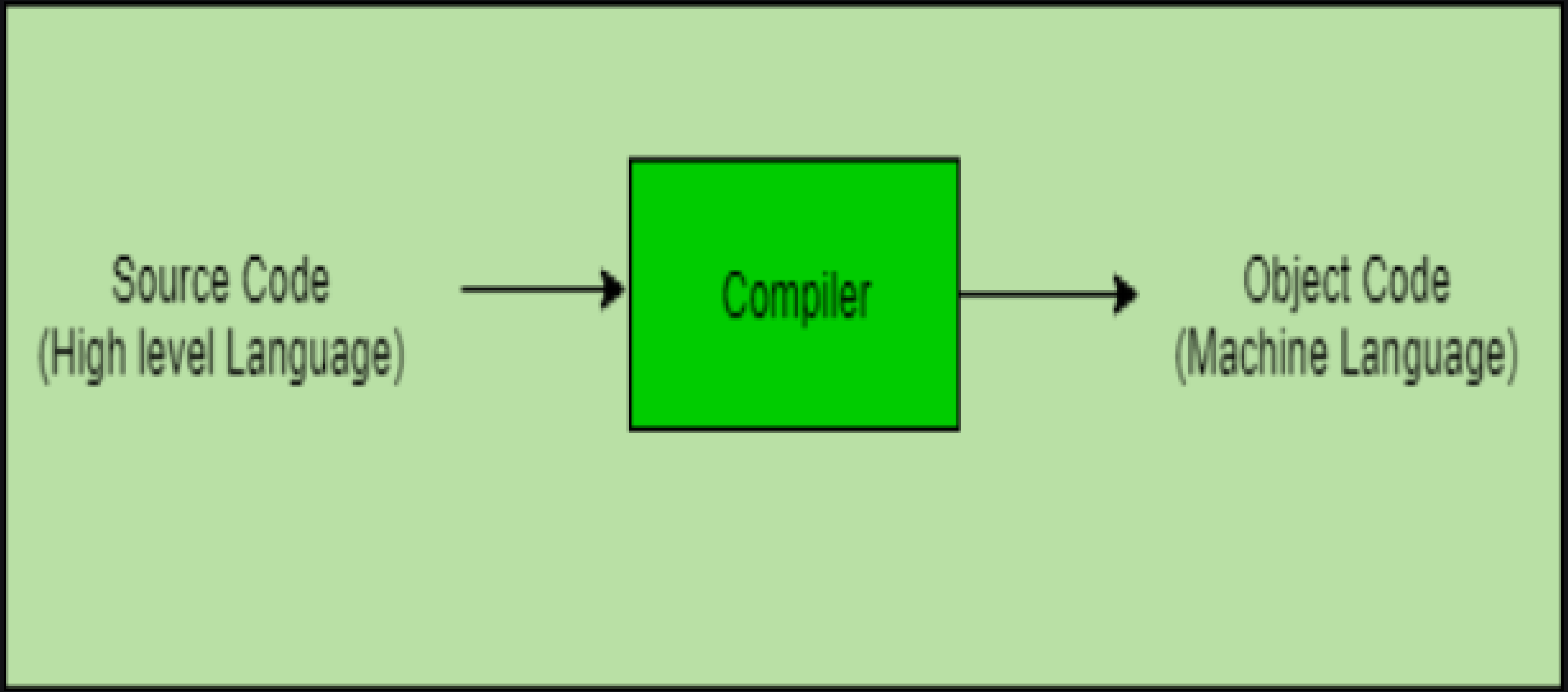
Visit www.pediaa.com

3 c

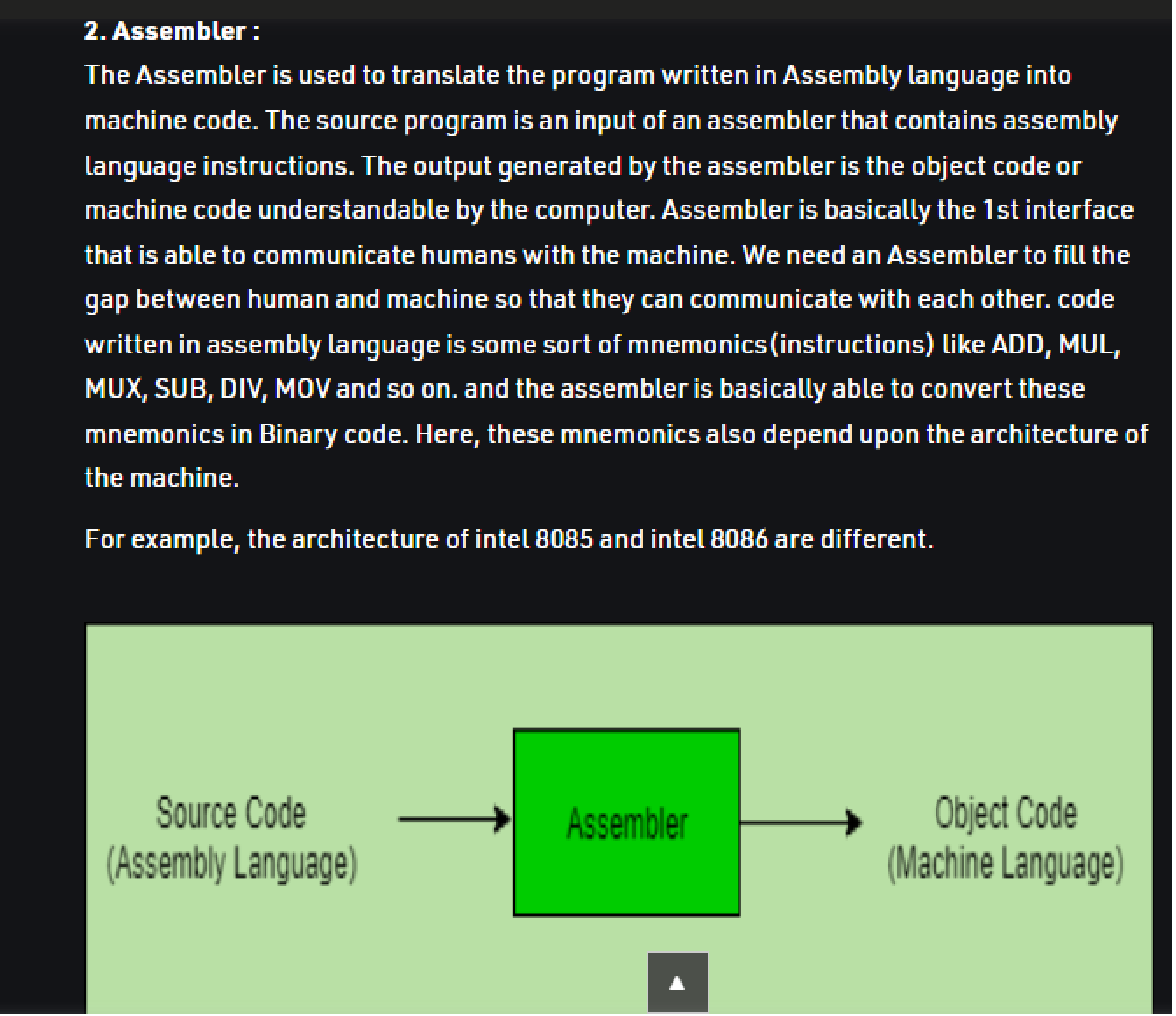
1. Compiler :

The language processor that reads the complete source program written in high-level language as a whole in one go and translates it into an equivalent program in machine language is called a Compiler. Example: C, C++, C#, Java.

In a compiler, the source code is translated to object code successfully if it is free of errors. The compiler specifies the errors at the end of the compilation with line numbers when there are any errors in the source code. The errors must be removed before the compiler can successfully recompile the source code again



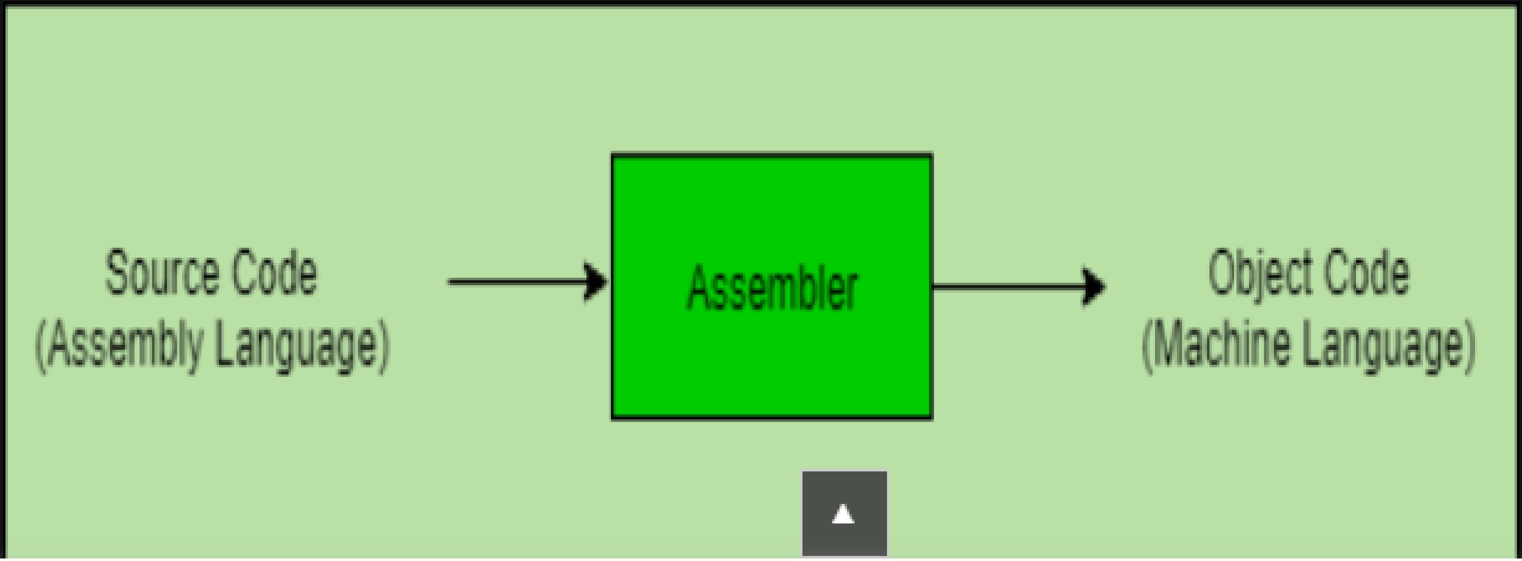
3 c



2. Assembler :

The Assembler is used to translate the program written in Assembly language into machine code. The source program is an input of an assembler that contains assembly language instructions. The output generated by the assembler is the object code or machine code understandable by the computer. Assembler is basically the 1st interface that is able to communicate humans with the machine. We need an Assembler to fill the gap between human and machine so that they can communicate with each other. code written in assembly language is some sort of mnemonics(instructions) like ADD, MUL, MUX, SUB, DIV, MOV and so on. and the assembler is basically able to convert these mnemonics in Binary code. Here, these mnemonics also depend upon the architecture of the machine.

For example, the architecture of intel 8085 and intel 8086 are different.

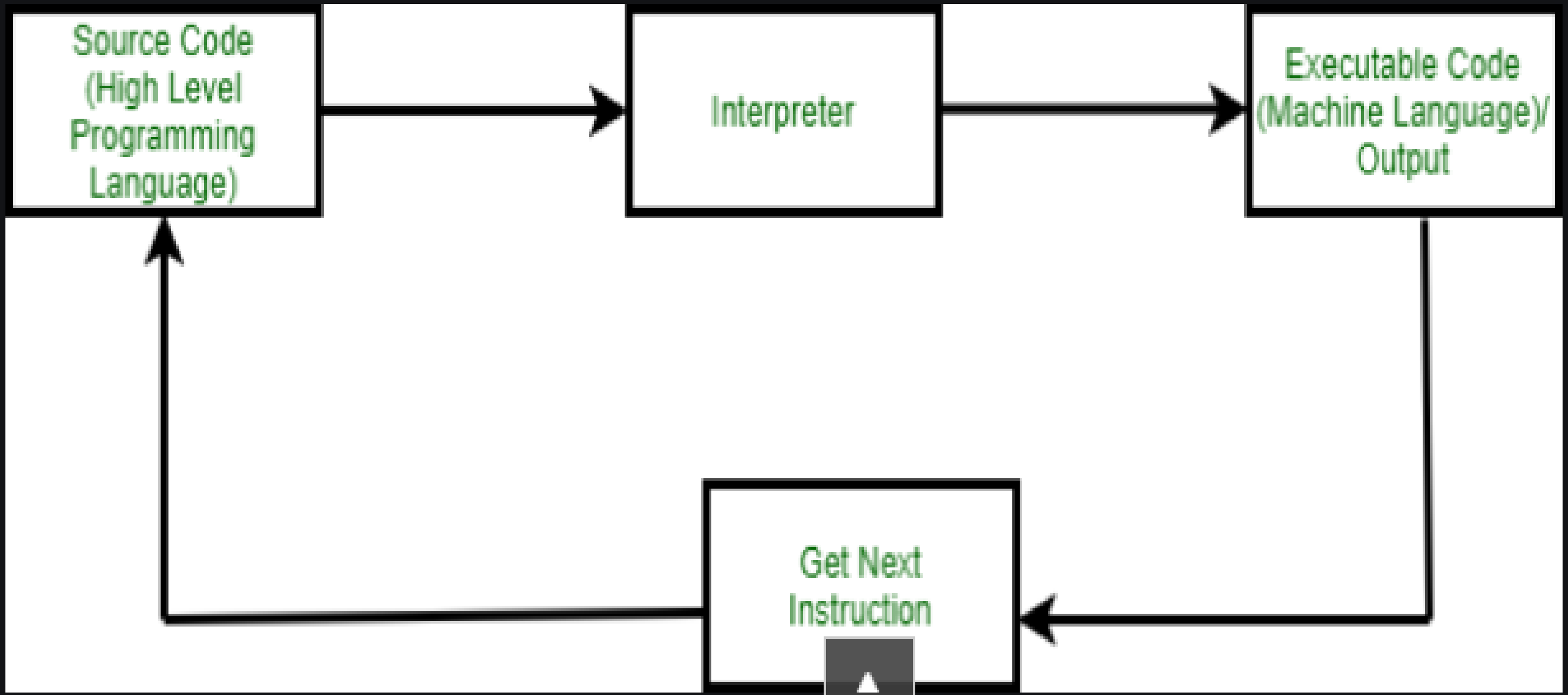


3 c

3. Interpreter :

The translation of a single statement of the source program into machine code is done by a language processor and executes immediately before moving on to the next line is called an interpreter. If there is an error in the statement, the interpreter terminates its translating process at that statement and displays an error message. The interpreter moves on to the next line for execution only after the removal of the error. An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code.

Example: Perl, Python and Matlab.


Source Code (High Level Programming Language)
Interpreter
Executable Code (Machine Language)/ Output
Get Next Instruction

Declaration of Variables

Variables are the basic unit of storage in a PL. These variables consist of a data type, the variable name, and the value to be assigned to the variable. Unless and until the variables are declared and initialized, they cannot be used in the program. Let us learn more about the Declaration and Initialization of Variables in this article below.

What is Declaration and Initialization?

•**Declaration** of a variable in a computer programming language is a statement used to specify the variable name and its data type. Declaration tells the compiler about the existence of an entity in the program and its location. When you declare a variable, you should also initialize it.

•**Initialization** is the process of assigning a value to the Variable. Every programming language has its own method of initializing the variable. If the value is not assigned to the Variable, then the process is only called a Declaration.

Rules to Declare and Initialize Variables

4 c

There are few conventions needed to be followed while declaring and assigning values to the Variables –

1.Variable names must begin with a letter, underscore, non-number character. Each language has its own conventions.

2.Few programming languages like PHP, Python, Perl, etc. do not require to specify data type at the start.

3.Always use the '=' sign to initialize a value to the Variable.

4.Do not use a comma with numbers.

5.Once a data type is defined for the variable, then only that type of data can be stored in it. For example, if a variable is declared as Int, then it can only store integer values.

6.A variable name once defined can only be used once in the program. You cannot define it again to store another type of value.

7.If another value is assigned to the variable which already has a value assigned to it before, then the previous value will be overwritten by the new value.

A **union** is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

5 a

Rest union vs structure 2017 Q (5-A)

Key Differences Between if-else and switch

1.The expression inside if statement decides whether to execute the statements inside if block or under else block. On the other hand, the expression inside a switch statement decides which case to execute.

2.You can have multiple if statement for multiple choice of statements. In switch, you only have one expression for the multiple choices.

3.If-else statement checks for equality as well as for logical expression. On the other hand, switch checks only for equality.

4.The if statement evaluates integer, character, pointer or floating-point type or boolean type. On the other hand, switch statement evaluates only character or an integer datatype.

5.Sequence of execution is like either statement under if block will execute or statements under else block statement will execute. On the other hand, the expression in switch statement decide which case to execute and if you do not apply a break statement after each case it will execute till the end of the switch statement.

6.If expression inside if turn outs to be false, statement inside else block will be executed. If expression inside switch statement turns out to be false then default statements is executed.

7.It is difficult to edit if-else statements as it is tedious to trace where the correction is required. On the other hand, it is easy to edit switch statements as they are easy to trace.

Definition of if-else

5 c

The if-else statements belong to selection statements in OOP. The general form of the if-else statements is as follow

```
if (expression){
statement(s)
}else{
statement(s)
}
```

where “if” and “else” are the keywords, and the statements can be a single statement or a block of statements. The expression evaluates to be “true” for any non-zero value and for zero it evaluates to be “false”.

Definition of switch

5 c

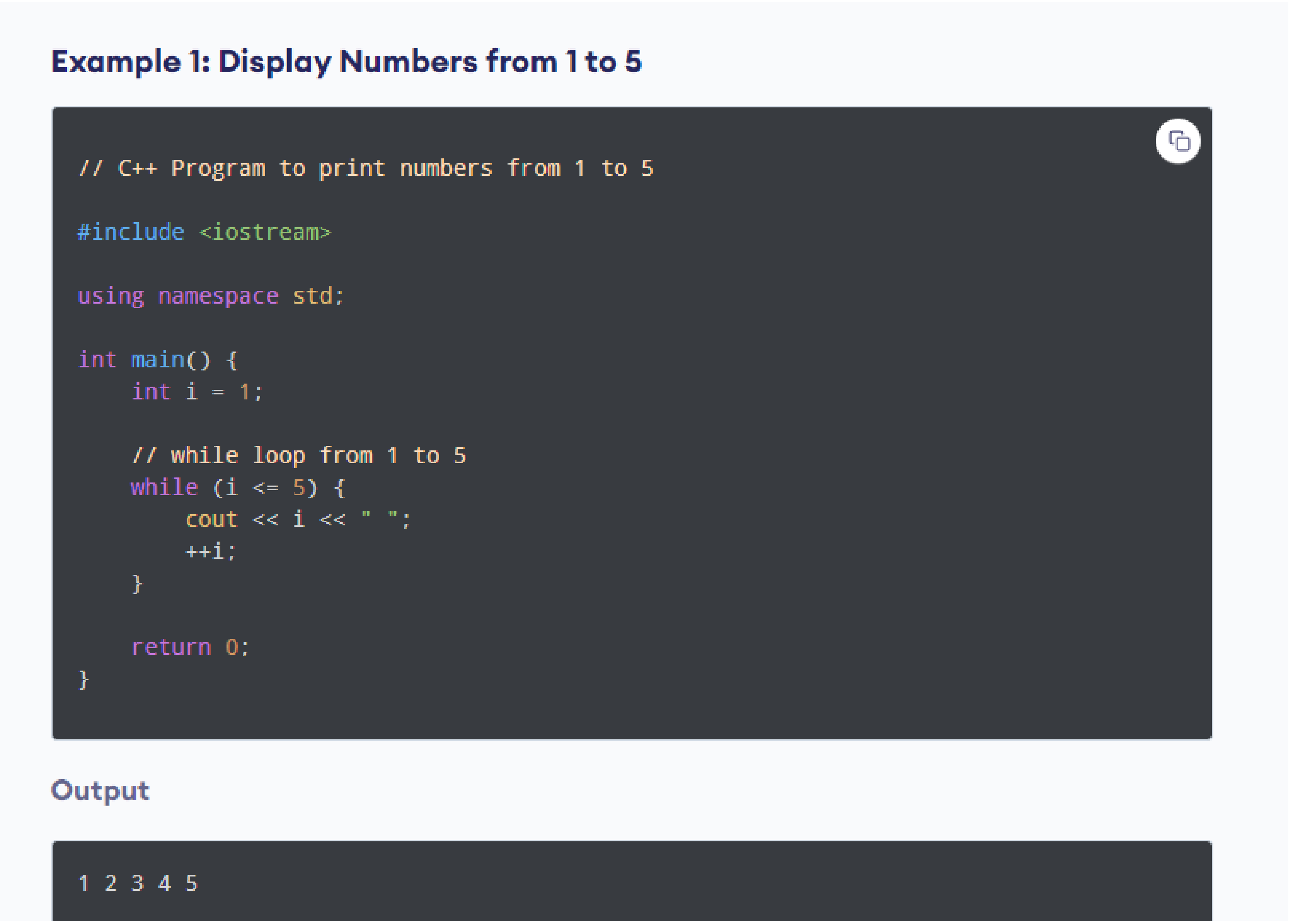
The switch statements is a multiple-choice selection statement. The general form of the switch statement is as follow

```
switch( expression ){
case constant1:
statement(s);
break;
case constant2:
statement(s);
break;
case constant3:
statement(s);
break;
.
.
default
statement(s);
}
```

Where the expression evaluates an integer or character constants. The expression here only evaluates for equality. The expression is verified against the constants present in the case statements. If a match is found, the statements associated with that case is executed, until a “break” occurs. As the break statement is optional in the case statements, if the break statement is not present then, the execution does not stop until the end of the switch statement.

At do while loop , the while portion is in the end of the loop. So no matter what happens the loop runs one time then go to the the condition part to verify.

7 c



Example 1: Display Numbers from 1 to 5

```
// C++ Program to print numbers from 1 to 5

#include <iostream>

using namespace std;

int main() {
    int i = 1;

    // while loop from 1 to 5
    while (i <= 5) {
        cout << i << " ";
        ++i;
    }

    return 0;
}
```

Output

```
1 2 3 4 5
```

7 c

Here is how the program works.

Iteration	Variable	i <= 5	Action
1st	`i = 1`	`true`	`1` is printed and `i` is increased to `2`.
2nd	`i = 2`	`true`	`2` is printed and `i` is increased to `3`.
3rd	`i = 3`	`true`	`3` is printed and `i` is increased to `4`
4th	`i = 4`	`true`	`4` is printed and `i` is increased to `5`.
5th	`i = 5`	`true`	`5` is printed and `i` is increased to `6`.
6th	`i = 6`	`false`	The loop is terminated

A **goto** statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function. **8 b**

NOTE – Use of **goto** statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

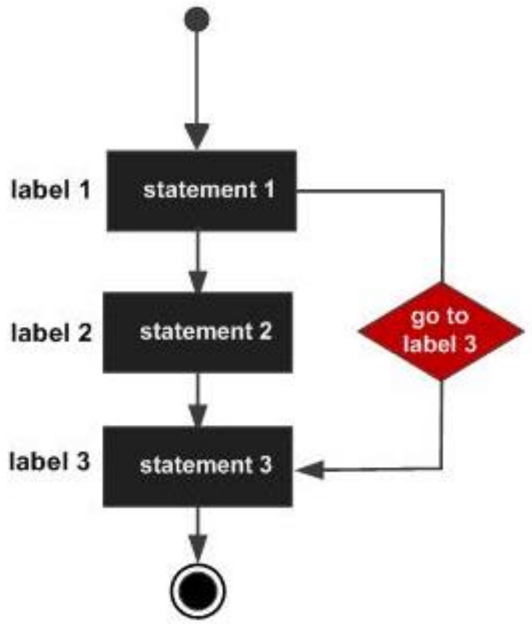
Syntax

The syntax for a **goto** statement in C is as follows –

```
goto label;
..
.
label: statement;
```

Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement.

Flow Diagram



8 b

Live Demo

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   LOOP:do {

      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }

      printf("value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

```
void recursion() {
   recursion(); /* function calls itself */
}

int main() {
   recursion();
}
```

The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

Advantages of recursion

1. The code may be easier to write.
2. To solve such problems which are naturally recursive such as tower of Hanoi.
3. Reduce unnecessary calling of function.
4. Extremely useful when applying the same solution.
5. Recursion reduce the length of code.
6. It is very useful in solving the data structure problem.
7. Stacks evolutions and infix, prefix, postfix evaluations etc.

Programmers use the **ternary operator** for decision making in place of longer **if** and **else** conditional statements.

The ternary operator take three arguments:

1. The first is a comparison argument
2. The second is the result upon a true comparison
3. The third is the result upon a false comparison

It helps to think of the ternary operator as a shorthand way or writing an if-else statement. Here's a simple decision-making example using **if** and **else**:

8 d

```
int a = 10, b = 20, c;

if (a < b) {
    c = a;
}
else {
    c = b;
}

printf("%d", c);
```

This example takes more than 10 lines, but that isn't necessary. You can write the above program in just 3 lines of code using a ternary operator.

Syntax

`condition ? value_if_true : value_if_false`

The statement evaluates to `value_if_true` if `condition` is met, and `value_if_false` otherwise.

Here's the above example rewritten to use the ternary operator:

```
int a = 10, b = 20, c;

c = (a < b) ? a : b;

printf("%d", c);
```

Output of the example above should be:

8 d

```
10
```

`c` is set equal to `a`, because the condition `a < b` was true.

Remember that the arguments `value_if_true` and `value_if_false` must be of the same type, and they must be simple expressions rather than full statements.

8 d

Ternary operators can be nested just like if-else statements. Consider the following code:

```
int a = 1, b = 2, ans;
if (a == 1) {
    if (b == 2) {
        ans = 3;
    } else {
        ans = 5;
    }
} else {
    ans = 0;
}
printf ("%d\n", ans);
```

Here's the code above rewritten using a nested ternary operator:

```
int a = 1, b = 2, ans;
ans = (a == 1 ? (b == 2 ? 3 : 5) : 0);
printf ("%d\n", ans);
```