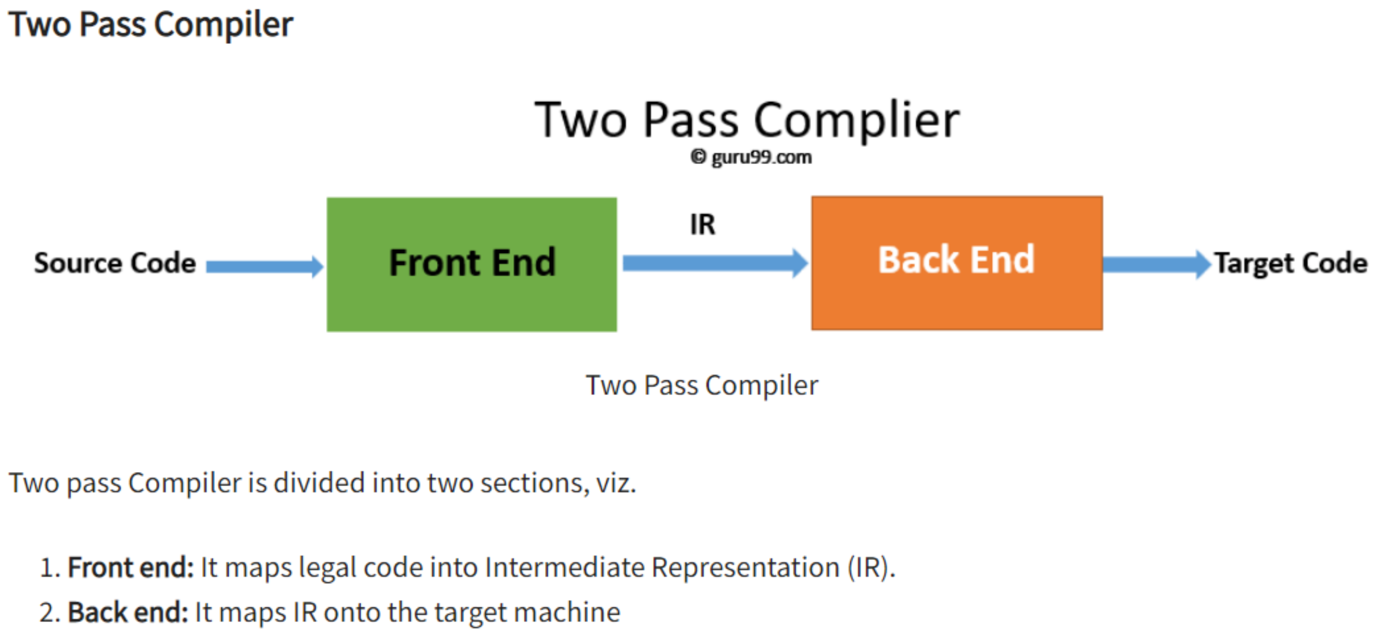
Tutorial 6 Answers – Made By U19CS012

1. Explain two-pass assembler functions with example program.



Two Pass Compiler

Two Pass Compiler

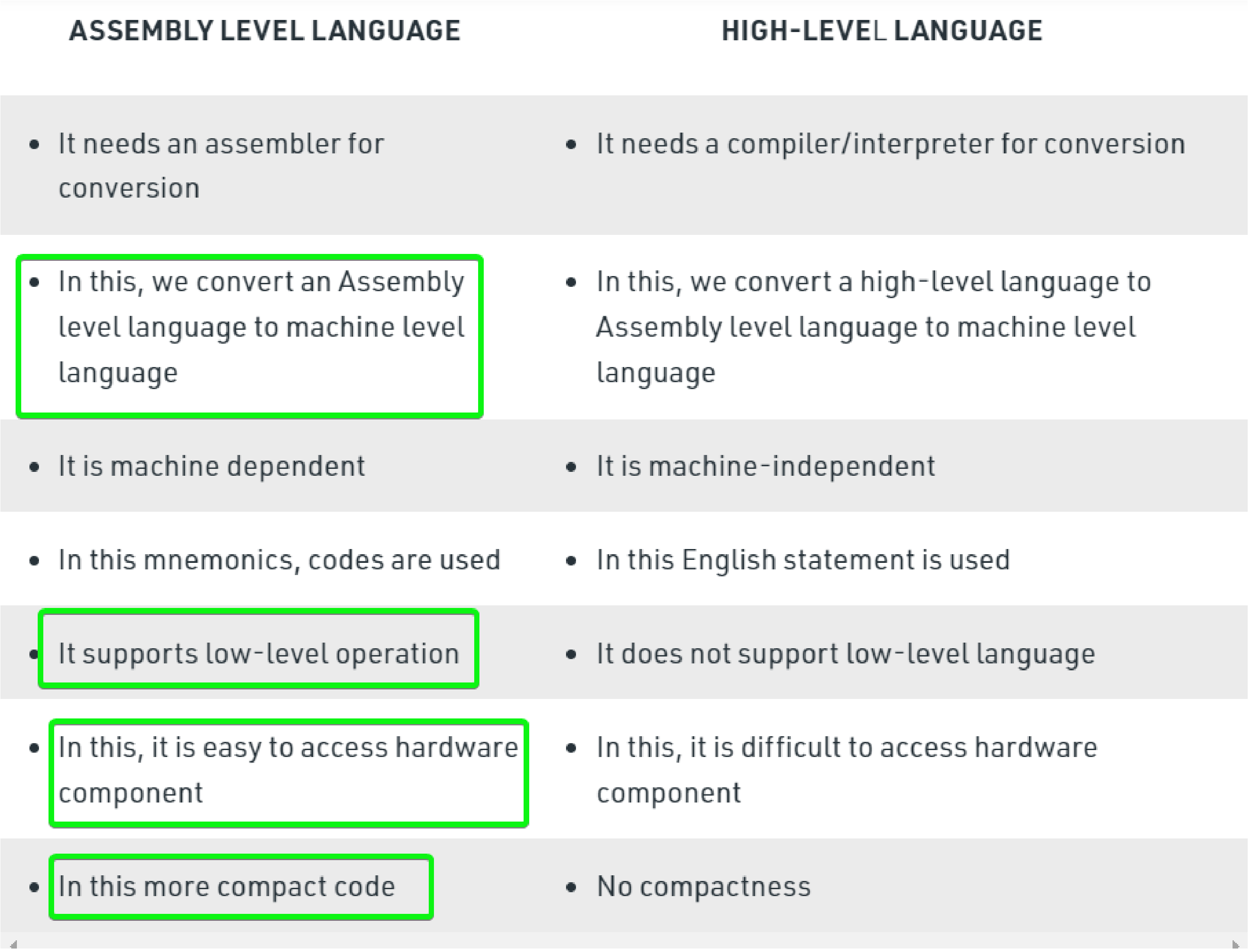
Two pass Compiler is divided into two sections, viz.

1. **Front end:** It maps legal code into Intermediate Representation (IR).
2. **Back end:** It maps IR onto the target machine

References

1) https://www.entcengg.com/two-pass-assemblers/

2. What are some advantages of assembly languages over high level languages?

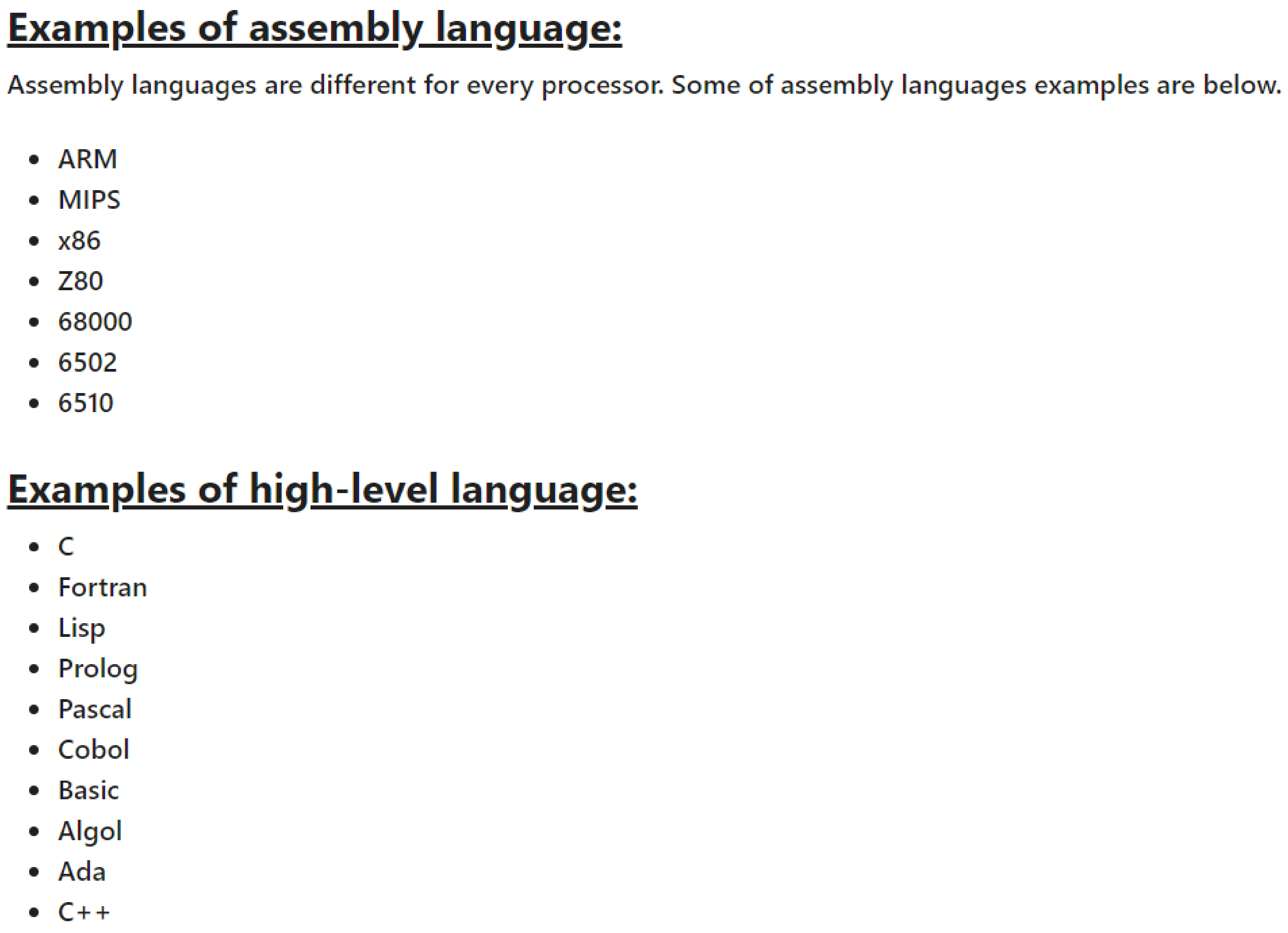


ASSEMBLY LEVEL LANGUAGE	HIGH-LEVEL LANGUAGE
• It needs an assembler for conversion	• It needs a compiler/interpreter for conversion
• In this, we convert an Assembly level language to machine level language	• In this, we convert a high-level language to Assembly level language to machine level language
• It is machine dependent	• It is machine-independent
• In this mnemonics, codes are used	• In this English statement is used
• It supports low-level operation	• It does not support low-level language
• In this, it is easy to access hardware component	• In this, it is difficult to access hardware component
• In this more compact code	• No compactness

Advantages

Below are the advantages:

1. It allows complex jobs to run in a simpler way.
2. It is memory efficient, as it requires less memory.
3. It is faster in speed, as its execution time is less.
4. It is mainly hardware-oriented.
5. It requires less instruction to get the result.
6. It is used for critical jobs.
7. It is not required to keep track of memory locations.
8. It is a low-level embedded system.



Examples of assembly language:

Assembly languages are different for every processor. Some of assembly languages examples are below.

- ARM
- MIPS
- x86
- Z80
- 68000
- 6502
- 6510

Examples of high-level language:

- C
- Fortran
- Lisp
- Prolog
- Pascal
- Cobol
- Basic
- Algol
- Ada
- C++

3. What tools are used for compiler construction?

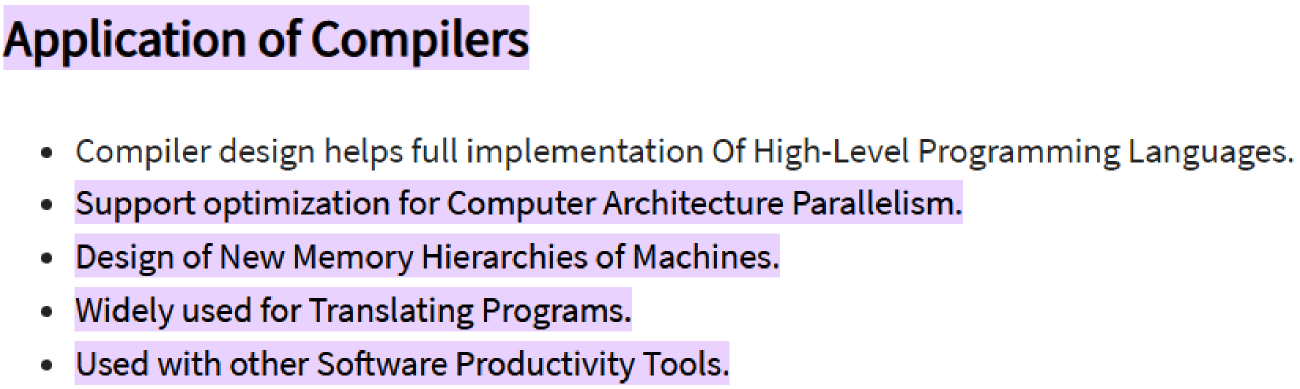
Parser generators - YACC

• Scanner generators – Lex

https://www.geeksforgeeks.org/compiler-construction-tools/

https://ecomputernotes.com/compiler-design/compiler-construction-tools

4. What are applications of compiler? Explain.

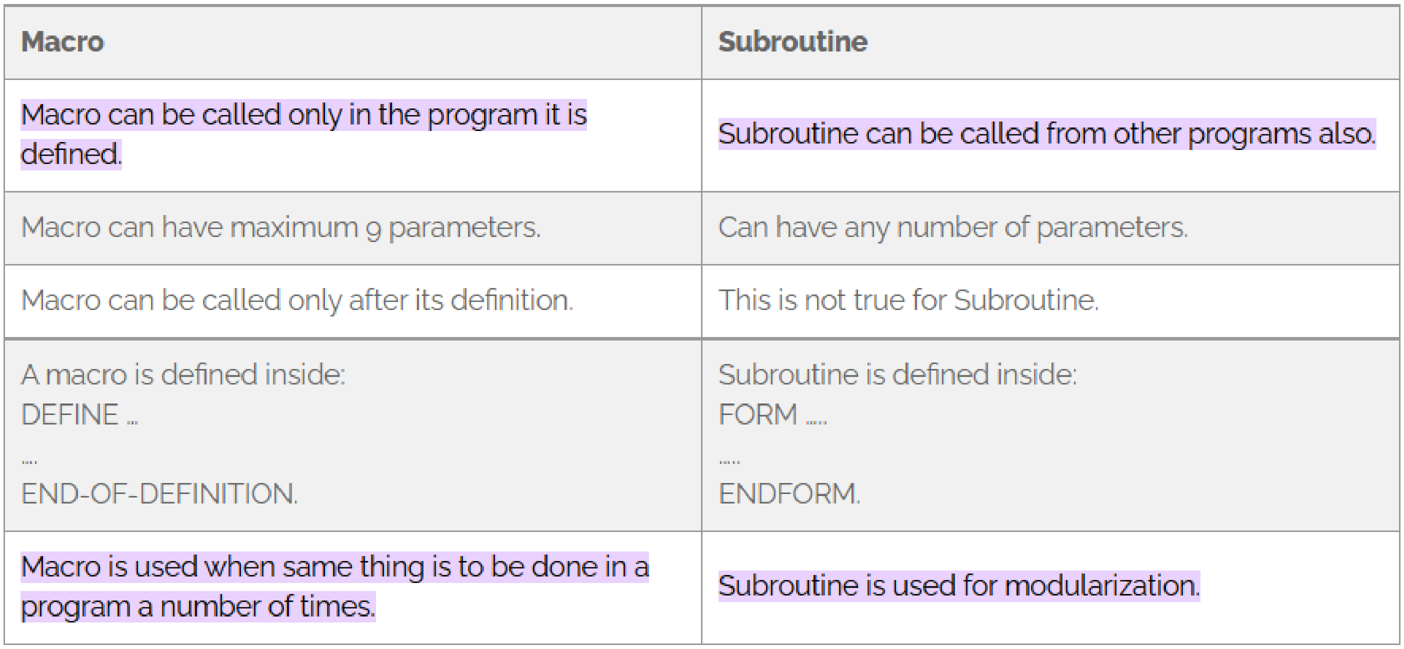


Application of Compilers

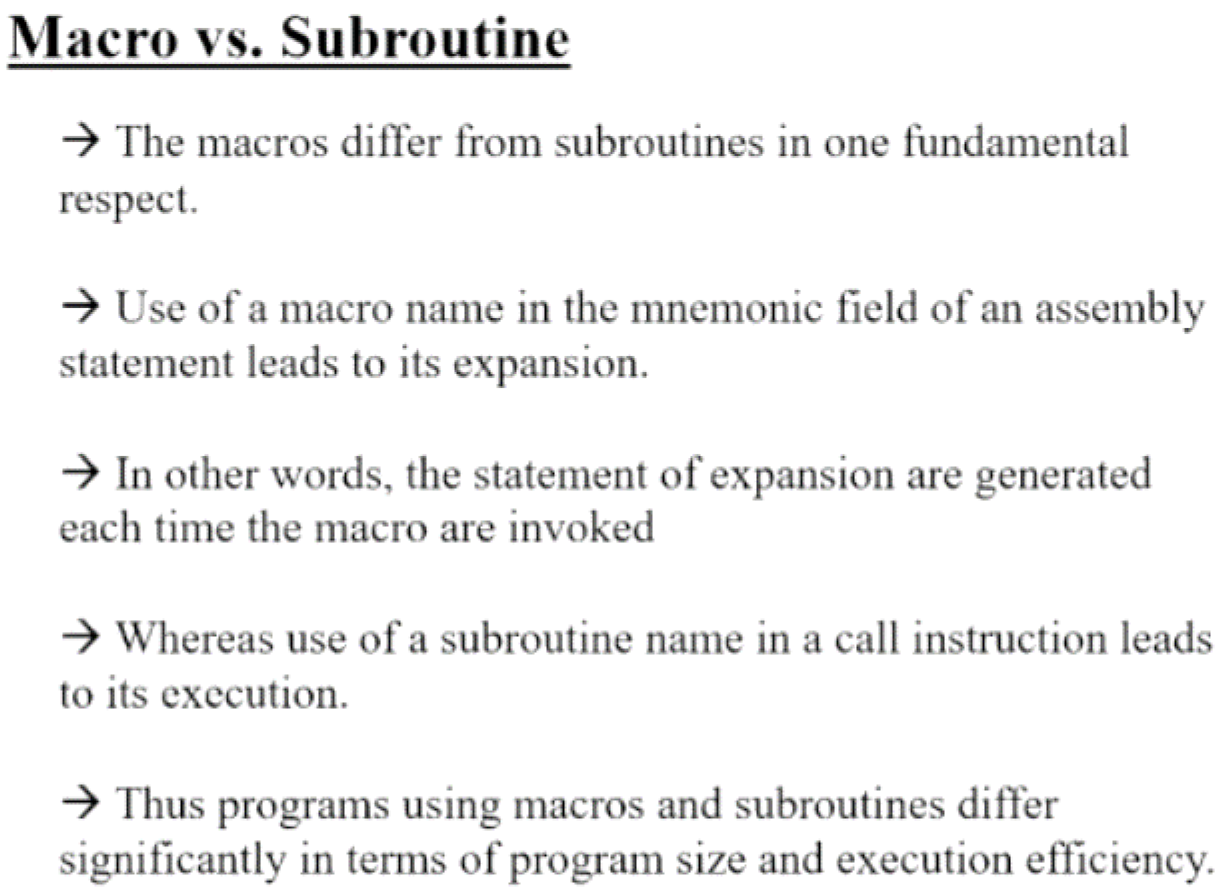
- Compiler design helps full implementation Of High-Level Programming Languages.
- Support optimization for Computer Architecture Parallelism.
- Design of New Memory Hierarchies of Machines.
- Widely used for Translating Programs.
- Used with other Software Productivity Tools.

https://www.geeksforgeeks.org/applications-of-compiler-technology/

5. Differentiate between a macro and a subroutine. And explain macro definition and expansion using an example.



Macro	Subroutine
Macro can be called only in the program it is defined.	Subroutine can be called from other programs also.
Macro can have maximum 9 parameters.	Can have any number of parameters.
Macro can be called only after its definition.	This is not true for Subroutine.
A macro is defined inside: DEFINE … …. END-OF-DEFINITION.	Subroutine is defined inside: FORM ….. ….. ENDFORM.
Macro is used when same thing is to be done in a program a number of times.	Subroutine is used for modularization.



Macro vs. Subroutine

→ The macros differ from subroutines in one fundamental respect.

→ Use of a macro name in the mnemonic field of an assembly statement leads to its expansion.

→ In other words, the statement of expansion are generated each time the macro are invoked

→ Whereas use of a subroutine name in a call instruction leads to its execution.

→ Thus programs using macros and subroutines differ significantly in terms of program size and execution efficiency.

https://www.geeksforgeeks.org/difference-between-macro-and-procedure/

https://www.geeksforgeeks.org/macros-and-its-types-in-c-cpp/

MACRO DEFINITION AND CALL

Macro definition: A macro definition is enclosed between a ***macro header statement*** and a ***macro end statement***.

→ Macro definitions are typically located at the start of the program.

→ A macro definition consists of :

1) A macro prototype statement.
2) One or more model statements.
3) Macro preprocessor statements.

→The macro prototype statement declares the name of a macro and the names and kinds of its parameters.

→A model statement is a statement from which an assembly language statement may be generated during macro expansion.

→A preprocessor statement is used to perform auxiliary functions during macro expansion.

→The macro prototype statement has the following syntax :

<macro name> [<formal parameter spec> [,..]]

where <macro name> appears in the mnemonic field of an assembly statement and <formal parameter spec> is of the form &<parameter name> [<parameter kind>]

Example showing the definition of macro INCR

```
MACRO
INCR          &MEM_VAL, &INCR-VAL, &REG
MOVER         &REG, &MEM_VAL
ADD           &REG, &INCR_VAL
MOVEM         &REG, &MEM_VAL
MEND
```

→MACRO and MEND are the macro header & macro end statements.

Macro expansion

→ A macro call leads to macro expansion.

→ During macro expansion, the macro call statement is replaced by a sequence of assembly statements.

→To differentiate between the original statements of a program and the statements resulting from macro expansion, each expanded statement is marked with a '+' preceding its label field.

Algorithm of macro expansion

```
1) MEC:=statement no of first statement following the prototype statement;

2) While statement pointed by MEC is not a MEND statement
    (a) if a model statement then
        (i) expand the statement
        (ii) MEC:=MEC+1;
    (b) else(i.e. a preprocessor statement)
        (i) MEC:=new value specified in the statement

3) Exit from macro expansion
```

Standard Definitation:

→A macro is a unit of specification for program generation through expansion.

→A macro consists of a name, a set of formal parameters and a body of code.

→The use of macro name with a set of actual parameters is replaced by some code generated by its body.

This is called **Macro Expansion**.

There are two kinds of expansions:
1) **Lexical expansion**
2) **Semantic expansion**

1) **Lexical expansion**: lexical expansion implies replacement of a character string by another character string during program execution.

→It is typically employed to replace occurrences of formal parameters by corresponding actual parameters.

2) **Semantic expansion**: semantic expansion implies generation of instructions tailored to the requirements of a specific usage—for eg : generation of type specific instructions for manipulation of byte and word operands.

→It is characterized by the fact that different uses of a macro can lead to codes which differ in the number, sequence and opcodes of instructions.