1. Explain the following terms and their usages

(a) compiler

(b) assembler

(c) linker

(d) loader

Sol:

https://en.wikipedia.org/wiki/Compiler

https://en.wikipedia.org/wiki/Assembly_language#Assembler

https://en.wikipedia.org/wiki/Linker_(computing)

https://en.wikipedia.org/wiki/Loader_(computing)

2. What are the stages a computer program undergoes, from initial creation to deployment and execution?

Sol:

https://en.wikipedia.org/wiki/Program_lifecycle_phase

3. Explain the purposes of the three steps “lexical analysis”, “parsing process” and “code generation” in the compilation process.

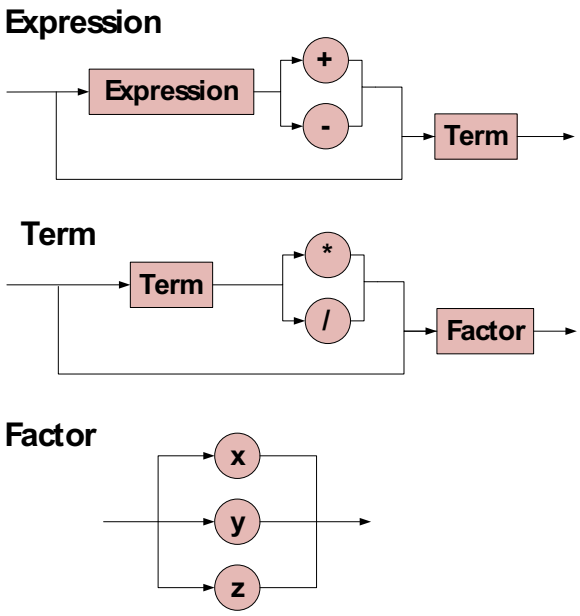
Sol:

- “lexical analysis” :
  - recognizing which strings of symbols from the source program represent a single entity called token
  - identifying whether they are numeric values, words, arithmetic operators, and so on.
- “parsing process” :
  - group tokens into statements based on a set of rules, collectively called a grammar
- “code generation”
  - constructing the machine-language instructions to implement the statements recognized by the parser and represented as syntax trees

4. Algebraic expressions manipulating variables x, y and z, such as “x-y*z+x/y”, can be described by the following grammar recursively:

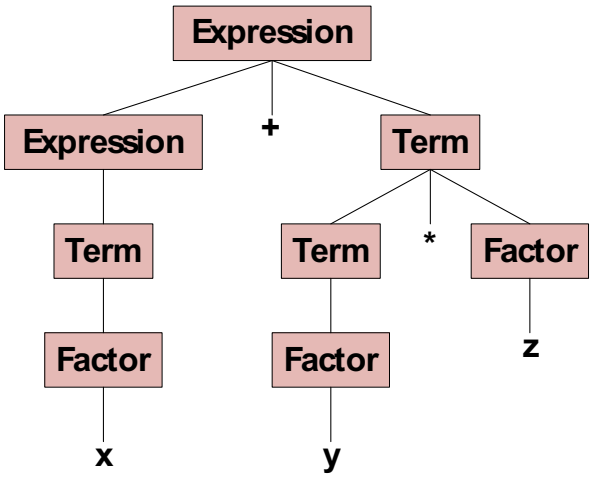
Expression := Term | Expression ADDSUB Term
Term := Factor | Term MULDIV Factor
Factor := x | y | z

where “|” means “or”, or equivalently by the following syntax diagrams



For the string **x + y * z**, draw the parse tree based on the above syntax diagrams.

Sol:

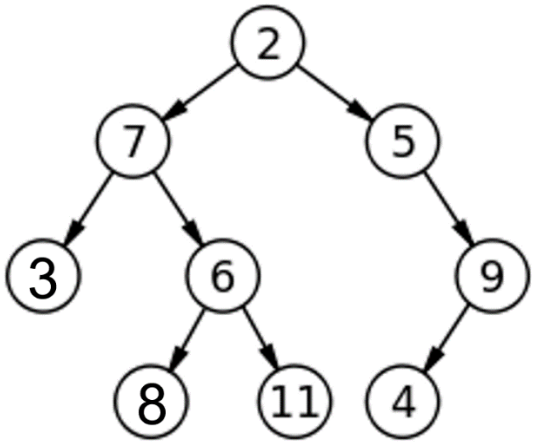


5. A *palindrome* is a string that reads the same forward and backward, such as **otto** or **madamimadam**. To make things simple, we shall consider describing only the palindromes with alphabet {0, 1}. This language includes strings like 0110, and 11011, but not 011 or 0101. Design a syntax diagram representing the grammatical structure of the palindromes with alphabet {0, 1}.

Sol:

P := empty | 0 | 1 | 0P0 | 1P1

6. Show the pre-order, in-order, and post-order traversal sequences of the following binary tree.



Sol:

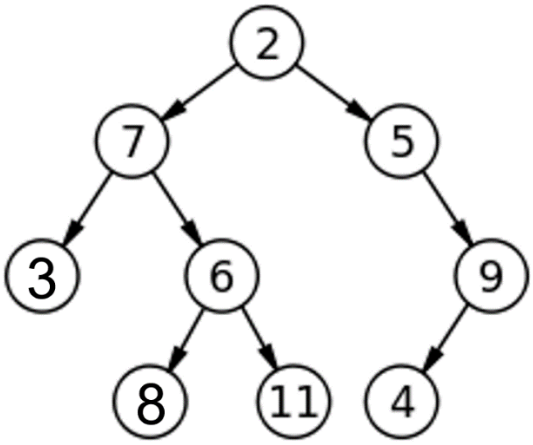
- Pre-order: 2 7 3 6 8 11 5 9 4
- In-order: 3 7 8 6 11 2 5 4 9
- Post-order: 3 8 11 6 7 4 9 5 2

7. Given the pre-order and in-order traversal sequences of a binary tree:

Pre-order: 2 7 3 6 8 11 5 9 4
In-order: 3 7 8 6 11 2 5 4 9,

show the structure of this binary tree.

Sol:



8. Consider the following code. Fill in the missing outputs (from ① to ⑪).

```
#include <stdio.h>
#include <stdlib.h>
typedef struct _Node {
    int val;
    struct _Node *next;
} Node;

int main(void)
{
    Node **pp;
    Node *p, *head = NULL;
    int i;

    printf("Part 1:\n");
    for (i=3; i>0; --i) {
        p = (Node*) malloc(sizeof(Node));
        p->val = i;
        p->next = head;
        head = p;
    }
    p = head;
```

```
    while (p) {
        printf("%p: %d | %p\n", p, p->val, p->next);
        p = p->next;
    }

    pp = &head;
    p = head;

    printf("Part 2:\n");
    printf("%p\n", p);
    printf("%p\n", *pp);

    while (p) {
        if (p->val == 2)
            *pp = p->next;

        pp = &p->next;
        p = p->next;
    }

    printf("Part 3:\n");
    p = head;
    while (p) {
        printf("%p: %d | %p\n", p, p->val, p->next);
        p = p->next;
    }
    return 0;
}
```

```
Part 1:
0xf74002e0: 1 | ①
0xf74002d0: 2 | ②
0xf7400080: 3 | ③
```

Part 2:

④

⑤

Part 3:

⑥: ⑦ | ⑧

⑨: ⑩ | ⑪

Sol:

Part 1:
0xf74002e0: 1 | 0xf74002d0
0xf74002d0: 2 | 0xf7400080
0xf7400080: 3 | 0x0
Part 2:
0xf74002e0
0xf74002e0
Part 3:
0xf74002e0: 1 | 0xf7400080
0xf7400080: 3 | 0x0