1. (12%) Explain the usages of following terms

(a) Linker

(b) Loader

(c) Compiler

2. (10%) Translate the statement "*a+=b*" into assembly code. Suppose integer variable *a* is in memory space 0X00F0 and integer variable *b* is in memory space 0X00F8. And there are 8 registers, r0-r7, to use. The available assembly instructions are listed in the table below.

Opcode	Operand1	Operand2	Meaning
MOV	Register1	Register2	Move data from register2 to register1
	Register1	Constant	Set the value of register1 constant
	Register1	[Addr2]	Move the data (4 bytes) in memory addressed Addr2 to register1.
	[Addr1]	Register2	Move the data (4 bytes) from register2 to the memory addressed Addr1.
ADD	Register1	Register2	Add the values in register1 to register2 and store the result in register1

3. (10%) Explain how CPU, memory and registers work together to execute the statement "*a += b*". You may use the translated assembly code in question 2 to illustration the steps.

4. (10%) Algebraic expressions manipulating variables x, y and z, such as "x-y*z+x/y", can be described by the following grammar recursively:

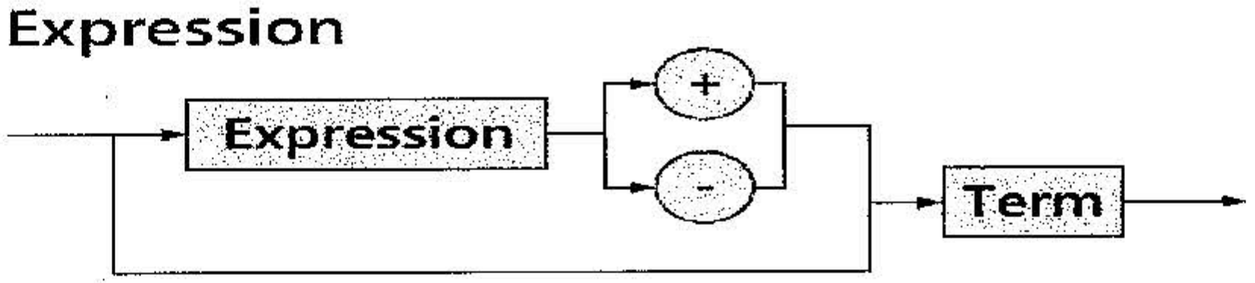
Expression := Term | Expression ADDSUB Term

Term := Factor | Term MULDIV Factor

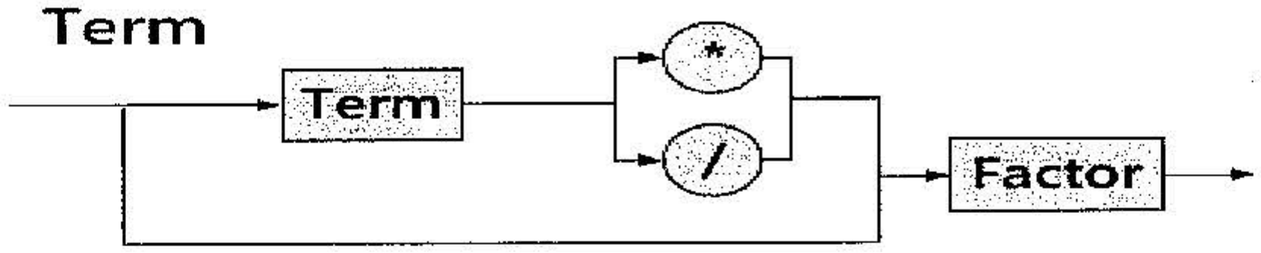
Factor := x | y | z

where "|" means "or", or equivalently by the following syntax diagrams

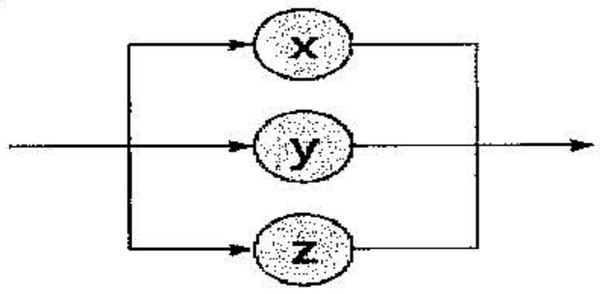
Expression



Term



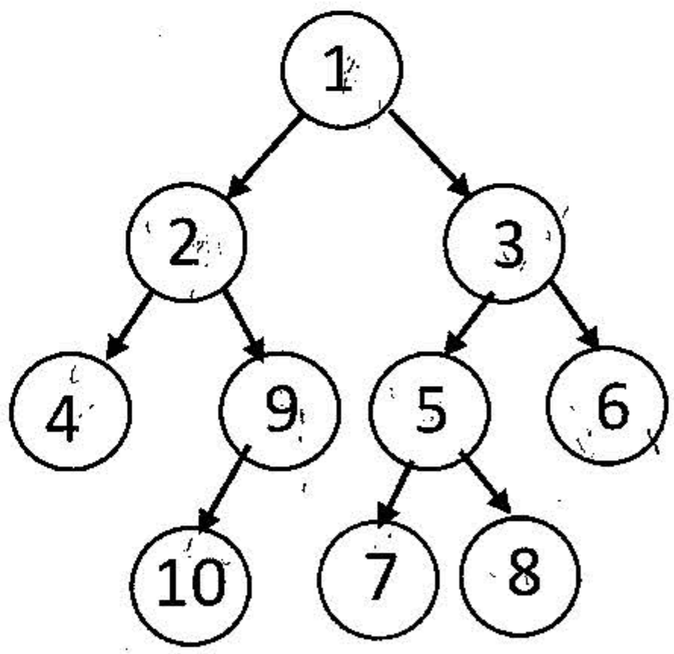
Factor



For the string **A * B + C * D**, draw the **parse tree** based on the above syntax diagrams.(Note: not syntax tree)

5. (10%) Please design a syntax diagram to describe the strings of the form ww^R, where w is any string consisting of ***a*** and ***b***, and w^R is w's reverse. (For example, ***abba***, ***bbaabb*** are possible sentence patterns)

6. (9%) Show the pre-order, in-order, and post-order traversal sequences of the following binary tree.



7. (9%) Given the pre-order and in-order traversal sequences of a binary tree:

preorder: 1 2 4 5 7 8 3 6 9 10

inorder: 4 2 7 5 8 1 3 9 6 10

show the structure of this binary tree.

8. (30%) Consider the following code. Fill in the missing outputs (from ① to ⑮).

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int val;
    struct Node *next;
} Node;

Node* operation1(Node* h, int v)
{
    Node *p;
    p = (Node *) malloc(sizeof(Node));
    p->val = v;
    p->next = h;
    return p;
}

Node* operation2(Node *p)
{
    Node *q = NULL, *r;
    while (p) {
        r = p->next;
        p->next = q;
        q = p;
        p = r;
    }
    return q;
}
void print(Node *p)
{
    while (p) {
        printf("%p: %d | %p\n", p, p->val, p->next);
        p = p->next;
    }
}

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

    for (i=0; i<3; i++) {
        head = operation1(head, i);
    }
    print(head);
    printf("============\n");

    head = operation2(head);
    print(head);

    return 0;
}
```

0x0800: ① | ②

0x07f0: ③ | ④

0x07e0: ⑤ | ⑥

==========

⑦ : ⑧ | ⑨

⑩ : ⑪ | ⑫

⑬ : ⑭ | ⑮