1. Please construct context-free grammars **with ε-free productions** for the following languages.
   1. {i|i∈N(Natural number), and i is a palindrome, and (i mod 5)=0}
   2. {ω| ω∈(a,b,c,d)\* and the numbers of a’s ,b’s and c’s occurred in ω are **even**, and ω **starts with a or c , ends with d** }
   3. {ω| ω∈(a,b,c,d)\* and the numbers of a’s and b’s and c’s occurred in ω are **odd**, and ω **starts with a, ends with c or d**}
   4. {ω| ω∈(a,b,c,d)\* and the numbers of a’s and b’s and c’s occurred in ω are **even**, and ω **starts with b or c, ends with a**}
   5. {ω| ω∈(a,b,c,d)\* and the numbers of a’s and b’s and c’s occurred in ω are **odd**, and ω **starts with a , ends with b or c**}
   6. L={ω| ω∈(a,b,c,d)\* and the numbers of a’s and b’s and c’s occurred in ω are **even**, and ω **starts with a or b** }
   7. L={ω| ω is a non-negative palindrome integer which is read the same backward as forward, and ω is an even number}
   8. L={anωbn| n≥0,ω∈(c,d,e,f)\* and the numbers of c’s and d’s and e’s occurred in ω are **even** }
2. Please construct a **DFA** **with minimum states** for the following regular expression.
3. (((a|b)\*a)\*(a|b))\*(a|b)
4. ((a|b)\*(a|b)(a|b)\*)\*(ab)\*(a|b)
5. (a|(a|(a|b\*))\*)\*(a|b\*)
6. ((a|b)\*|(ab)\*)\*(a|b)
7. (((a|b)\*(ab)(a|b)\*)\*(ab)(a|b)\*)\*(a|b)
8. a(a(a|b)\*b)\*(a|b)\*(a|b)b
9. ((a|b)\*|(ab)\*)\*(ab)(a|b)
10. ((aa\*|bb\*)\*(a|b))\*(a|b\*)
11. Please **eliminate the left recursions (if there are)** and **extract maximum common left factors (if there are)** from the following context free grammar, and then decide **the resulted grammar** is whether a LL(1) grammar by **constructing the related LL(1) parsing table.**(20%)

S → i E t S | i E t S e S | a

E → E and F | F

F → F or G | G

G → b

P → b S d

S → S ; A | A

A → B | C

B → a

C → D | D e A

D → E B

E → i F t

F → F o G | G

G → b

S → if E then S | if E then S else S | a

E → E and F | F

F → F or G | G

G → ( E ) | b

S → begin L end | if E then S | if E then S else S | while E do S | a

L → L ; S | S

E → E and E | E or E | b

S → if E then S | if E then S else S | while E do S | id = F

E → E and E | E or E | not E | ( E ) | b

F → F + F | F \* F | ( F ) | n

S → begin L end | if E then S | if E then S else S | while E do S | a

L → L ; S | S

E → E or F | F

F → F and G | G

G → ( E ) | b

A → B a | A a | c

B → B b | A b | d

S → S a | A b

A → A c | B d

B → S e | f | C g

C → B h | j

1. Please **construct a LR(1) parsing table for the following ambiguous grammar with the additional conditions** that
2. all θi (i=1,2) has the properties of right associative law, and θ2 has lower precedence than θ1.

E → E θ1 E | E θ2 E | ( E ) | i

1. \*, ⊗ and ⊕ have the properties of left associative law, and \* has higher precedence than ⊗, ⊗ has higher precedence than ⊕.

E → E ⊕ E | E ⊗ E | E \* | ( E ) | a | b

1. Please **construct a LR(1) parsing table for the following ambiguous grammar with your own defined additional conditions (You determine the required additional conditions by yourself)**.(15%)
2. S → S a S | S b S | c S d | e S | f
3. S → if E then S | if E then S else S | a

E → E and E | E or E | b

1. E → E θ1 E | E θ2 E | E θ3 E | ( E ) | id
2. S → if E then S | if E then S else S | a

E → E and E | E or E | ( E ) | b

1. S → if C then S | if C then S else S | A

C → b

A → d = E

E → E + E | E \* E | i

1. E → E and E | E or E | not E | ( E ) | b
2. Please show that the following operator grammar is whether an operator precedence grammar by **constructing the related parsing table.** (10%)

E → E a F | F

F → F o T | T

T → ( E ) | n T | b

1. Please show that **if a grammar G is a LL(1) grammar, then G must be a LR(1) grammar (**20%):
2. Please construct **an annotated parse tree** for the input string where the syntax-directed definition is as following (10%):
3. 123.123

Productions Semantic Rules

S→L(1).L(2)  S.val=L(1).val+L(2).val/4L(2).len

S→L S.val=L.val

L→L(1)B L.val=L(1).val\*4+B.val, L.len=L(1).len+1

L→B L.val=B.val, L.len=1

B→0 B.val=0

B→1 B.val=1

B→2 B.val=2

B→3 B.val=3

1. 101.101

Productions Semantic Rules

S→L1.L2 {S.val=L1.val+L2.val/2L2.len}

S→L {S.val=L.val}

L→L1B {L.val=L1.val\*2+B.val, L.len=L1.len+1}

L→Q {L.val=B.val,L.len=1}

B→0 {B.val=0}

B→1 {B.val=1}

1. (2+3\*4+@5)+@6\*7

Productions Semantic Rules

E→E1\*F {E.val=E1.val\*F.val}

E→F {E.val=F.val}

F→F1+G {F.val=F1.val+G.val}

F→G {F.val=G.val}

G→(E) {G.val=E.val}

G→i {G.val=i.lexval}

G→@i {G.va l=0-i.lexval}

1. 4\*5+6

Productions Semantic Rules

E→E1\*T E.val=E1.val\*T.val

E→T E.val=T.val

T→T1+F T.val=T1.val+F.val

T→F T.val=F.val

F→i F.val=i.**lexval**

1. 4+@(5\*@6+7)+8\*9

Productions Semantic Rules

E→E1\*T E.val=E1.val\*T.val

E→T E.val=T.val

T→T1+F T.val=T1.val+F.val

T→F T.val=F.val

F→(E) F.val=E.val

F→@F1 F.val=0-F1.val

F→i F.val=i.**lexval**

1. We assume that the storage organization and the form of activation record used in C language program run-time stack storage allocation are as following. Please **construct the run-time stack map when it gets the maximum size at the third time** for the following C program (10%).

**Notes:** Here we assume that the caller’s sp of Main function is the start address of global variable data area, and the returned address in the activation record of a function (including Main function) is filled by the operating system automatically, you might not care it.

1. The initial value of variable X is 6, the start address of stack used in the program is K. The stack map may get its maximum size for several times, here we ask you draw the stack map at maximum size for the third time.

#include <stdio.h>

int x, y;

int main()

{

x = 6;

y = f(x);

}

int f(int n)

{

if (n <= 1)

return 1;

else if (n == 2)

return 2;

else {

int t1, t2, t3, t;

t1 = f(n - 1);

t2 = f(n - 2);

t3 = f(n - 3);

t = t1 + t2;

t = t + t3;

return t

}

}

1. The initial value of variable X is 10 and the initial value of variable Y is 6, the start address of stack used in the program is K.

#include <stdio.h>

int x, y, z;

int main()

{

x = 10;

y = 6;

z = C(x, y);

}

int C(int m, int n)

{

if (m >= n)

if (n <= 0) return 1;

else if (n == 1) return m;

else {

int t1;

t1 = m - n;

if (t1 < n) {

int t2;

t2 = C(m, t1);

return t2

} else {

int t3, t4, t5, t6, t7;

t3 = m - 1;

t4 = n - 1;

t5 = C(t3, t4);

t6 = m \* t5;

t7 = t6 / n;

return t7;

}

} else return 0;

}

1. The initial value of variable X is 10, the start address of stack used in the program is K. The stack map may get its maximum size for several times, here we ask you draw the stack map at maximum size for the second time.

#include <stdio.h>

int x, y;

int main()

{

    x = 10;

    y = f(x);

}

int f(int m)

{

    if (m >= 0)

        if (m == 0) return(1)

        else if (m == 1) return(1)

        else if (m == 2) return(2)

        else {

            int t1, t2, t3, t;

            t1 = f(m - 1);

            t2 = f(m - 2);

            t3 = f(m - 3);

            t = t1 + t2 + t3;

            return(t)

        } else return(-1)

}

1. The initial value of variable X is 10, the start address of stack used in the program is K. The stack map may get its maximum size for several times, here we ask you draw the stack map at maximum size for the second time.

#include <stdio.h>

int x, y;

int main()

{

    x = 10;

    y = f(x);

}

int f(int n)

{

    if (n <= 1)

        return 1;

    else if (n == 2)

        return 2;

    if (n == 3)

        return 3;

    else {

        int t1, t2, t3, t4, t5, t6, t;

        t1 = f(n - 1);

        t2 = f(n - 2);

        t3 = f(n - 3);

        t4 = f(n - 4);

        t5 = t1 + t2;

        t6 = t3 + t4;

        t = t5 + t6;

        return t

    }

}

1. Please translate the following program fragment **using short circuit code and back-patching techniques**.
2. Here we assume that the declaration of array A is array [1..10,1..10], each data element of array A would **use 4 storage unit,** and the start address of array A’s storage area is addrA.

i = 1;

m = 0;

loop = 0;

n = 0;

while (loop == 0 && i <= 10) {

j = 1;

while (loop == 0 && j <= i)

if (a[i, j] != a[j, i]) {

loop = 1;

m = i;

n = j;

} else j = j + 1;

if (loop == 0) i = i + 1;

}

1. Here we assume that the declaration of array A and B are array [1..10,1..20], each data element of array A or B would **use 4 storage unit,** and the start address of array A’s storage area is addrA, the start address of array B’s storage area is addrB.

i = 1;

loop = 0;

while (loop == 0 && i <= 10) {

j = 1;

while (loop == 0 && j <= 20)

if (a[i, j] != b[i, j]) {

loop = 1;

m = i;

n = j;

} else j = j + 1;

if (loop == 0) i = i + 1;

};

if (loop == 1) e = 0;

else e = 1;

1. Here we assume that the declaration of array A, array B and array C are array [1..10,1..10], each data element of **array A,array B and array C would use 4 storage unit**, and the start address of array A’s storage area is addrA, the start address of array B’s storage area is addrB, the start address of array C’s storage area is addrC.

i = 1;

while (i <= 10) {

    j = 1;

    while (j <= 10) {

        k = 1;

        m = 0;

        while (k <= 10) {

            if (A[i, k] == 0 || B[k, j] == 0)

                k = k + 1;

            else {

                m = m + A[i, k] \* B[k, j];

                k = k + 1

            }

            C[i, j] = m;

        }

        j = j + 1;

    }

    i = i + 1

}

1. Here we assume that the declarations of array A,B,C are array [1..10,1..10], each data element of array A,B,C would **use 4 storage unit,** and the start address of array A’s storage area is addrA, the start address of array B’s storage area is addrB, the start address of array C’s storage area is addrC.

i = 1;

while (i <= 10) {

    j = 1;

    while (j <= 10) {

        c[i, j] = 0;

        j = j + 1

    }

    i = i + 1;

}

i = 1;

while (i <= 10) {

    j = 1;

    while (j <= 10) {

        k = 1;

        while (k <= 10) {

            if (a[i, k] != 0 && b[k, j] != 0)

                c[i, j] = c[i, j] + a[i, k] \* b[k, j];

            k = k + 1;

        }

        j = j + 1;

    }

    i = i + 1;

}

9. Please **construct the DAG** for the following basic block. Please optimize the block and **rewrite the block** in optimized code form.(10%)

1. We assume that only **variable P be used later**.

E=A+B

F=E-C

G=F\*D

H=A+B

I=H-C

L=I+G

M=I\*I

M=2\*M

N=L+M

P=N+M

1. We assume **only Variable L would be used later**

B=3

D=A+C

E=A\*C

F=D+E

G=B\*F

H=A+C

I=A\*C

J=H+I

K=B\*5

L=K+J

M=L

1. We assume that only variable “M” is live on exit.

C=A+B

D=A-B

E=A\*C

D=D\*E

F=A+B

F=F\*E

G=D\*E

M=F\*G