Principles of Compiling TEST

1. language→grammar with ε-free productions

test31

Please construct context-free grammars **with ε-free productions** for the following language (10%).

{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**}

test33

Please construct context-free grammars **with ε-free productions** for the following language (10%).

{ω| ω∈(a,b,c)* and the numbers of a's and b's and c's occurred in ω are **even**, and ω **starts with b , ends with a or c**}

test43

Please construct context-free grammars **with ε-free productions** for the following languages (20%).

(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** }

test61

Please construct context-free grammars **with ε-free productions** for the following language (10%).

{$a^m \omega b^m$| m≧0 and ω∈(c,d,e,f)* and the numbers of d's and e's and f's occurred in ω are **even**}

test65

Please construct context-free grammars **with ε-free productions** for the following language (10%).

{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**}

test71

Please construct **context-free grammars without ε -productions** for the following language.

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** }

2. DFA

test31 test33

Please construct a **DFA with minimum states** for the following regular expression. (10%)

(a|(a|(a|b*))*)*(a|b*)

test43

Please construct a **DFA with minimum states** for the following regular expression. (20%)

(((a|b)*a)*(a|b))*(a|b)

test61

Please construct a **DFA with minimum states** for the following regular expression. (15%)

(((a|b)*(ab)(a|b)*)*(ab)(a|b)*)*(a|b)

test65

Please construct a **DFA with minimum states** for the following regular expression. (15%)

((a|b)*(a|b)(a|b)*)*(ab)*(a|b)

test71

Please construct a **DFA with minimum states** for the following regular expression. (15%)

a(a(a|b)*b)*(a|b)*(a|b)b

3. the related LL(1) parsing table

test31 **test33**

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.**(15%)

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

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

test43

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→iEtS|iEtSeS|a

E→E and F|F

F→ F or G|G

G→b

test61

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.**(15%)

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

test65

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.**(15%)

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

test71

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.**(15%)

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

4. LR(1)

test31 test33

Please **construct a LR(1) parsing table for the following ambiguous grammar**

with the additional conditions that ***,** ⊗ and ⊕ **have the properties of left associative law**, and *** has higher precedence than** ⊗, ⊗ **has higher precedence than** ⊕.(15%)

E→E⊕E|E⊗**E|E*|(E)|a|b**

test43

Please **construct a LR(1) parsing table for the following ambiguous grammar with the additional conditions** that **all** θ_i **(i=1,2) has the properties of right associative law**, and θ_2 **has lower precedence than** θ_1.(20%)

E→E θ_1 E| E θ_2 E |(E)|i

Please show that **if a grammar G is a LL(1) grammar, then G must be a LR(1) grammar (**20%):

test61

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%)

E→ E θ_1 E|E θ_2 E| E θ_3 E| (E)|id

test65

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%)

S→SaS|SbS|cSd|eS|f

test71

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%)

S→if E then S| if E then S else S|a

E→ E and E|E or E|(E)|b

5. annotated parse tree

test31 test33

Please construct **an annotated parse tree** for the input string 123.123 where the syntax-directed definition is as following (10%):

Productions	Semantic Rules
$S \rightarrow L^{(1)}.L^{(2)}$	$S.val=L^{(1)}.val+L^{(2)}.val/4^{L(2).len}$
$S \rightarrow L$	$S.val=L.val$
$L \rightarrow L^{(1)}B$	$L.val=L^{(1)}.val*4+B.val$, $L.len=L^{(1)}.len+1$
$L \rightarrow B$	$L.val=B.val$, $L.len=1$
$B \rightarrow 0$	$B.val=0$
$B \rightarrow 1$	$B.val=1$

$B \rightarrow 2$	B.val=2
$B \rightarrow 3$	B.val=3

test61

Please construct **an annotated parse tree** for the input string (2+3*4+@5)+@6*7 where the syntax-directed definition is as following (10%):

Productions	Semantic Rules
$E \rightarrow E_1*F$	{E.val=E_1.val*F.val}
$E \rightarrow F$	{E.val=F.val}
$F \rightarrow F_1+G$	{F.val=F_1.val+G.val}
$F \rightarrow G$	{F.val=G.val}
$G \rightarrow (E)$	{G.val=E.val}
$G \rightarrow i$	{G.val=i.lexval}
$G \rightarrow @i$	{G.val=0-i.lexval}

test71

Please construct **an annotated parse tree** for the input string 4*5+6 where the syntax-directed definition is as following (10%):

Productions	Semantic Rules
$E \rightarrow E_1*T$	E.val=E_1.val*T.val
$E \rightarrow T$	E.val=T.val
$T \rightarrow T_1+F$	T.val=T_1.val+F.val

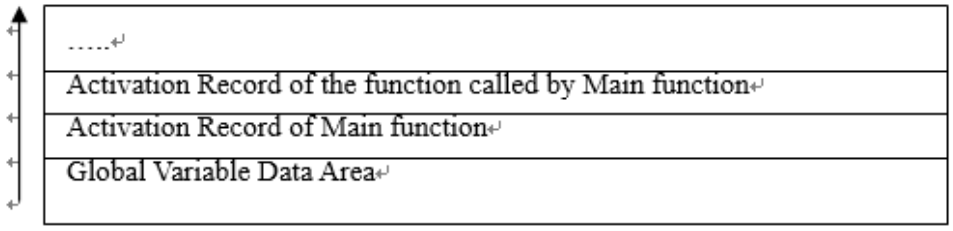
T→F	T.val=F.val
F→i	F.val=i.**lexval**

6. the run-time stack map

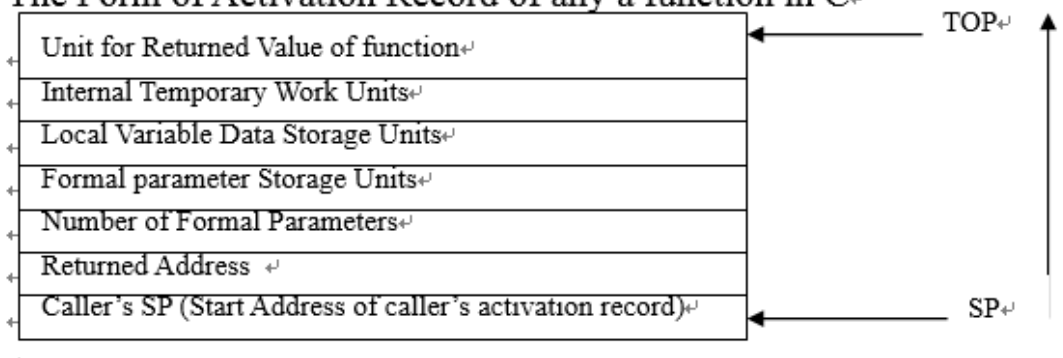
test31 test33

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 second time for the following C program (10%).

Storage Organization of C Language



The Form of Activation Record of any a function in C



The C program is as the following:

```
#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
}
  }
```

Notes: 1) 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.

2) The initial value of variable X is 6, the start address of stack used in the program is K.

3) 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

test61

```
#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)

}
```

Notes: 1) 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.

2) The initial value of variable X is 10, the start address of stack used in the program is K.

3) 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.

7. quadruple sequence

test31

Please translate the following program fragment into quadruple sequence. (10%)

```
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;
}
```

Notes: 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.

test33

Please translate the following program fragment into three address code sequence, divide the TAC sequence into basic blocks, construct the flow graph and find out all back edges in the flow graph. (20%)

```
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;
}
```

Notes: 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.

test61

Please translate the following program fragment into **Quadruple sequence using short circuit code and back-patching techniques**. (15%)

```
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
}
```

Notes: 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.

2) The related semantic rules are described as followings, where NXINSTR means "No. of Next Instruction" , the No. of first instruction is (1).

E →i {E•TC=NXINSTR; E•FC=NXINSTR+1;

GEN('if' i.place '<>0' 'goto 0'); GEN('goto 0')}

$E \rightarrow E_a$ rop E_b {E•TC=NXINSTR; E•FC=NXINSTR+1;

GEN('if' E_a.place rop.op E_b.place 'goto 0'); GEN('goto 0')}

$E \rightarrow (E^{(1)})$ {E•TC= $E^{(1)}$•TC; E•FC= $E^{(1)}$•FC}

$E \rightarrow$not $E^{(1)}$ {E•TC= $E^{(1)}$•FC; E•FC= $E^{(1)}$•TC}

$E^A \rightarrow E^{(1)}$ and {BACKPATCH($E^{(1)}$•TC,NXINSTR); E^A•FC= $E^{(1)}$•FC;}

$E \rightarrow E^A E^{(2)}$ {E•TC= $E^{(2)}$•TC; E•FC=MERG(E^A•FC,$E^{(2)}$•FC}

$E^0 \rightarrow E^{(1)}$ or {BACKPATCH($E^{(1)}$•FC,NXINSTR); E^0•TC= $E^{(1)}$•TC;}

$E \rightarrow E^0 E^{(2)}$ {E•FC= $E^{(2)}$•FC; E•TC=MERG(E^0•TC,$E^{(2)}$•TC}

$C \rightarrow$if E then {BACKPATCH(E•TC,NXINSTR); C•CHAIN=E•FC;}

$T \rightarrow C\ S^{(1)}$ else {q=NXINSTR; GEN('goto 0');

BACKPATCH(C•CHAIN,NXINSTR);

T •CHAIN=MERG($S^{(1)}$•CHAIN,q)}

$S \rightarrow T\ S^{(2)}$ {S•CHAIN=MERG(T•CHAIN,$S^{(2)}$•CHAIN)}

$S \rightarrow C\ S^{(1)}$ {S•CHAIN=MERG(C•CHAIN,$S^{(1)}$•CHAIN)}

$W \rightarrow$while {W•LABEL=NXINSTR}

$W^d \rightarrow W$ E do {BACKPATCH(E•TC,NXINSTR);W^d•CHAIN=E•FC;

W^d•LABEL=W•LABEL;}

$S \rightarrow W^d\ S^{(1)}$ {BACKPATCH($S^{(1)}$•CHAIN, W^d•LABEL);

GEN('goto' W^d •LABEL); S • CHAIN= W^d•CHAIN}

$A \rightarrow S$; {BACPATCH(S•CHAIN ,NXINSTR)}

$S \rightarrow AS^{(1)}$ {S•CHAIN = $S^{(1)}$•CHAIN }

(NXINSTR means "No. of Next Instruction")

8. DAG

test31

Please construct the DAG for the following basic block. We assume that only variable P be used later, please optimize the block and rewrite the block in optimized code form.(10%)

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

test61

Please **construct the DAG** for the following basic block, optimize the block and **rewrite the block** in optimized code form. Note that we assume **only Variable L would be used later**(10%)

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