中 国 科 学 技 术 大 学

2008－2009 学年第二学期考试试卷（A）

考试科目：编译原理和技术　　得分：_____

学生所在系：_____ 姓名：_____ 学号：_____

1、（15 分）直接画出接受正规式 $a^*b^*c^*$所表示语言的最简 DFA。

2、（5 分）用长度不超过 4 的某个句型的两个不同最左推导来说明下面的文法是二义的。

$S \rightarrow A\,S \mid b$

$A \rightarrow S\,A \mid a$

3、（15 分）证明下面的文法

$S \rightarrow S\,A \mid A$

$A \rightarrow a$

是 SLR(1)文法，但不是 LL(1)文法。

4、（15 分）教材第 206 页图 7.6（2003 年版第 221 页图 7.5）的翻译方案不适用于允许过程递归的语言。请修改此翻译方案，使得它适应允许直接递归过程的语言。不得改文法，通过增加文法符号的属性，调整和修改语义动作来达到目的（对那些语义动作维持不变的产生式及其语义动作不用写出）。

5、（5 分）教材第 377 页（2003 年版第 343 页）上说：

类 Point 的方法 translate 翻译成函数 translate_ _5Pointdd：

```
void translate_ _5Pointdd(Point &this, double x_offset , double y_offset) {
    this.xc += x_offset;    this.yc += y _offset;
}
```

其中假设 C 语言也有引用调用方式（形式参数前面加字符&）。请按 C 语言实际只有值调用方式来考虑，写出方法 translate 翻译成的 C 函数，并说明调用点需要做的修改。

6、（15 分）下面左右两边分别是两个 C 程序文件 file1.c 和 file2.c 的内容，用命令 gcc file1.c file2.c 对这两个文件进行编译和连接。请回答：

（1）编译器是否会报错？若你认为会，则说明理由。

（2）若编译器不报错，连接器是否会报错？若你认为会，则说明理由。

（3）若上面 2 步都不报错，则运行时是否会报错？若你认为会，则说明理由。

（4）若上面 3 步都不报错，则运行输出的结果是什么？说明理由。

```
char k = 2;
char j = 1;
```

```
#include <stdio.h>
extern short k;
main(){
    printf("%d\n", k);
```

```
}
```

7、（10 分）下面是一个以函数作为参数的 C 语言程序。

```
int f(int g()) {return g(g);}
main(){f(f);}
```

（1）对于函数类型的形式参数，调用时的参数传递传什么？

（2）该程序执行时，系统报告 Segmentation fault，请回答是什么原因。

8、（10 分）下面的 C 程序在运行时，系统报告 Segmentation fault，请回答是什么原因。

```
char s[10] = “123456789”;
char *p = “123456789”;
main() {
    *(s+1) = ‘3’;
    *(p+1) = ‘3’;
}
```

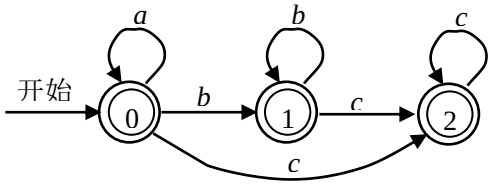
9、（10 分）下面左栏的 C 语言程序计算 5 的阶乘，结果是 120。经优化编译器生成的汇编程序跟在源程序的后面（从左栏看向右栏），其中在省略号处略去了 main 函数的汇编程序。请问优化编译器对 factorial 函数进行了怎样的优化。

```
factorial(long n, long res) {
    if (n==0) return res;
    else return factorial(n-1, n*res);
}
main() {
    printf("%d\n", factorial(5,1));
}
汇编程序：
    .file  "recursion.c"
    .text
    .p2align 4,,15
.globl factorial
    .type    factorial, @function
factorial:
    pushl    %ebp
    movl     %esp, %ebp
    movl     8(%ebp), %edx
    movl     12(%ebp), %eax
    testl    %edx, %edx
    je       .L3
    .p2align 4,,7
.L6:
    imull    %edx, %eax
    subl     $1, %edx
    jne      .L6
.L3:
    popl     %ebp
    ret
    .size    factorial, .-factorial
    … …          … …
    .ident   "GCC: (GNU) 4.2.3 (Debian 4.2.3-5)"
    .section .note.GNU-stack,"",@progbits
```

2008－2009 学年第二学期

编译原理和技术参考答案（A）

1、接受正规式 $a^*b^*c^*$所表示语言的最简 DFA 如下：



2、句型 *aSAS* 的两个不同最左推导如下：

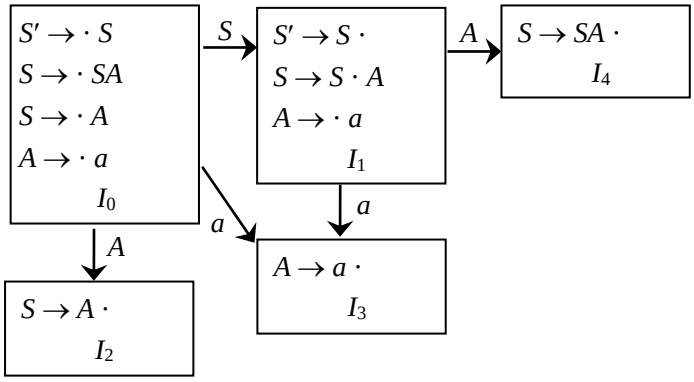
$S \Rightarrow AS \Rightarrow aS \Rightarrow aAS \Rightarrow aSAS$

$S \Rightarrow AS \Rightarrow SAS \Rightarrow ASAS \Rightarrow aSAS$

3、该文法的第一个产生式表现出直接左递归，因此该文法不是 LL(1)。

接受该文法的活前缀的 DFA 见下面右边；Follow(*S*′) = {$}，Follow(*S*) = {$, *a*}，Follow(*A*) = {$, *a*}；SLR(1)分析表见下面左边。该表无冲突，所以该文法是 SLR(1)的。

状态	动作		转移	
	a	$	*S*	*A*
0	*s*3		1	2
1	*s*3	*acc*		4
2	*r*2	*r*2		
3	*r*3	*r*3		
4	*r*1	*r*1		



4、若出现直接递归时，递归过程调用处会报告标识符没有声明。需要让递归过程名早一点进入符号表，仅第 4 和第 6 条产生式的语义动作有变化。

$P \rightarrow M\ D\ ;\ S$ {*addWidth*(*top*(*tblptr*), *top*(*offset*)); *pop*(*tblptr*); *pop*(*offset*);}

$M \rightarrow \varepsilon$ {*t* = *mkTable*(*nil*); *push*(*t*, *tblptr*); *push*(0, *offset*);}

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow$ **proc id** ; {*N.lexeme* = **id**.*lexeme*}*N* D_1 ; *S*

{*addWidth*(*top*(*tblptr*), *top*(*offset*)); *pop*(*tblptr*); *pop*(*offset*);}

$D \rightarrow$ **id** : *T* {*enter*(*top*(*tblptr*), **id**.*lexeme*, *T.type*, *top*(*offset*));

top(*offset*) = *top*(*offset*) + *T.width*;}

$N \rightarrow \varepsilon$ {*t* = *mkTable*(*top*(*tblptr*)); *enterProc*(*top*(*tblptr*), *N.lexeme*, *t*);

push(*t*, *tblptr*); *push*(0, *offset*);}

5、方法 translate 翻译成的 C 函数如下：

```
void translate_ _5Pointdd(Point * this, double x_offset , double y_offset) {
    this->xc += x_offset;    this->yc += y _offset;
```

}

在调用点，第 1 个实在参数前面应该加上取地址记号&。

6、（1）编译器不会报错，因为两个文件是分别编译的，因而不会发现其中的类型错误。

（2）连接器不会报错，因为可重定位代码中没有变量的类型信息。

（3）运行时也不会报错。

（4）若机器的特点是低地址放整数的低位，高地址放整数的高位，如 X86，则结果是 258，因为分配给变量 j 的字节正好作为变量 short k 的高位字节。否则结果是 513。

7、（1）函数作为参数时，只需要传递函数代码的入口地址。

（2）该程序会不断地递归调用，导致因活动记录栈溢出而报错。

8、程序中第 2 个常量串“123456789”存放在只读数据区，第 2 个赋值语句试图修改只读数据区的内容，因此系统报错。

9、使用了尾递归优化。对于尾递归调用，运行时不用建立新的活动记录，就在当前活动记录完成计算。针对这个例子叙述具体步骤如下：

（1）计算调用的实在参数 n-1 和 n*res，仍存放在相应的寄存器中（eax 存放函数返回值）。

（2）转到本函数第一条语句的起始地址继续执行（注意，不是本函数的入口地址，因为函数入口是一段调用序列代码，然后才是第一条语句的代码）。