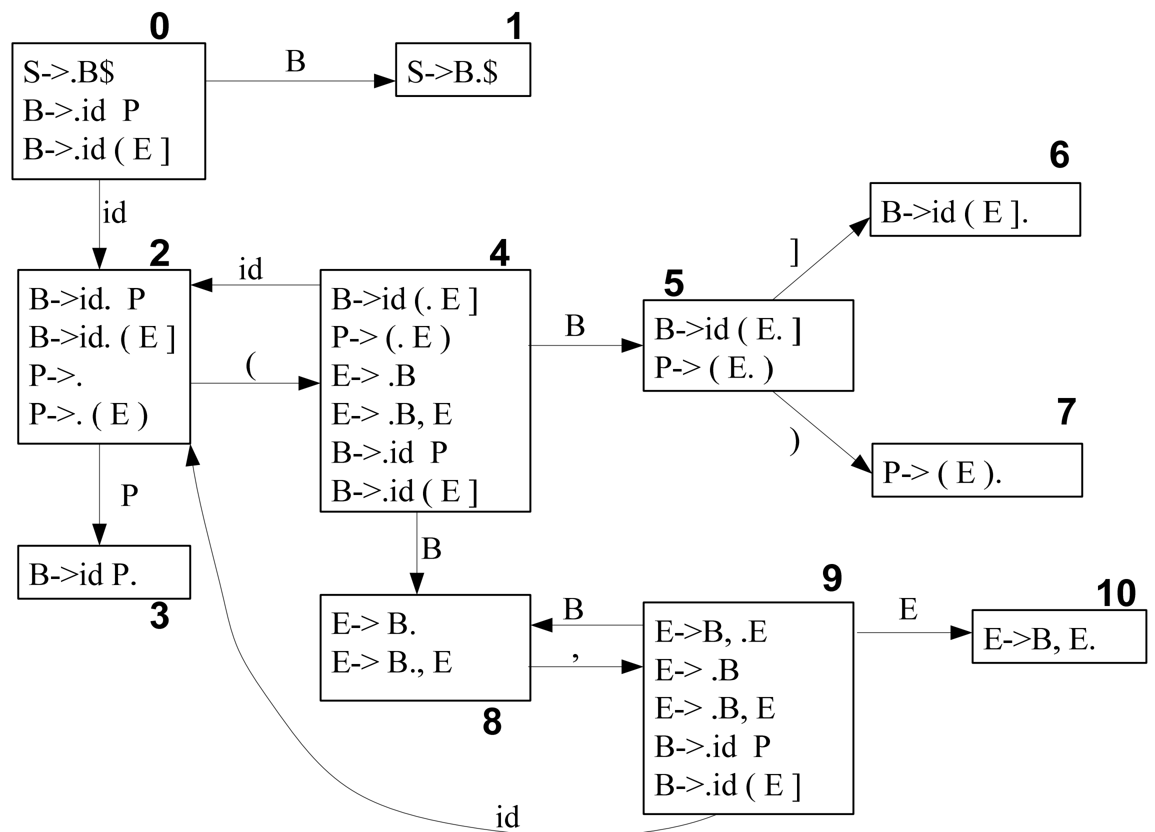
Homework 4
posted Feb. 26, due Mar. 10 at the beginning of the class (no late turn-ins will be accepted).
Please put a cover page over the homework. The grade will be written on the second page.

1. (30 pts) Exercise 3.11



	id	(	,	)	]	$	B	D	E
0	S2								
1						accept			
2	R3	R3,S4	R3	R3	R3	R3		G3	
3	R1	R1	R1	R1	R1	R1			
4	S2						G8		G5
5				S7	S6				
6	R2	R2	R2	R2	R2	R2			
7	R4	R4	R4	R4	R4	R4			
8	R5	R5	R5,S9	R5	R5	R5			
9	S2						G8		G10
10	R6	R6	R6	R6	R6	R6			

The grammar is SLR because for the cells with multiple options (e.g. R3,S4 in the cell < 2, ‘(‘ >), the multiple options can be distinguished by looking at the FOLLOW set . For example, ‘(‘ is not in FOLLOW(P) so that R3 should not be in the cell < 2, ‘(‘ >. In other words, when the state is 2, and the next input is ‘(‘, we should not reduce using rule P->Є because ‘(‘ is not in FOLLOW(P).

2. (30 pts) Given the following abstract grammar

   E-> E+E | E-E | E*E | id | num

   a. (5 pts) Design a concrete grammar for LL parsing.
   b. (10 pts) What is the AST for the input of “x-y*10”
   c. (5 pts) Write classes that represent ASTs.
   d. (10 pts) Use the visitor pattern to write an evaluator that evaluates the value of an expression. Write the classes on the paper.

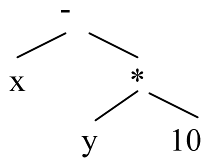
   a. E-> FE’
      E’->+E | -E | Є
      F -> T F’
      F’ -> *F | Є

T-> id | num

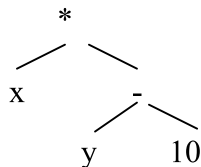
Note that another possible concrete grammar is as follows.
E->TE'
E'->+E | -E | *E | Є
T-> id | num

The first one is better because precedence is being considered.

b.



Note that your AST should be consistent with your concrete grammar. For instance, if you choose to have the second concrete grammar and the input is "x*y-10", you will have the following AST whose precedence is problematic. But as long as they are consistent, it shall be fine.



c.

```
public abstract class Exp {
    int accept(Visitor v);
}

public class PlusExp {
   private Exp e1, e2;
   public PlusExp (Exp a, Exp b) {
      e1=a;
      e2=b;
   }
    int accept(Visitor v) { return v.visit(this);}
}

public class MinusExp {
```

```
    private Exp e1, e2;
    public MinusExp (Exp a, Exp b) {
        e1=a;
        e2=b;
    }
     int accept(Visitor v) { return v.visit(this);}
}

public class TimesExp {
    private Exp e1, e2;
    public TimesExp (Exp a, Exp b) {
        e1=a;
        e2=b;
    }
     int accept(Visitor v) { return v.visit(this);}
}

public class IdExp {
    private String f0;
    public IdExp (String n0) { f0=n0; }
    int accept(Visitor v) { return lookup(f0);}

}

public class NumExp {
    private int n;
    public NumExp (int x) { n=x; }
    int accept(Visitor v) { return n;}

}
```

d.

```
 public EV extends visitor {
    public int visit (PlusExp e) {
        return e.e1.accept(this)+ e.e2.accept(this);
    }

    public int visit (MinusExp e) {
        return e.e1.accept(this)- e.e2.accept(this);
    }
    public int visit (TimesExp e) {
        return e.e1.accept(this)* e.e2.accept(this);
    }

    public int visit (IdExp e) {
         return e.accept(this);
```

```
    }

    public int visit (NumExp e) {
        return e.accept(this);
    }
}
```

3. (10 pts) Exercise 6.1 b and c.
   b. A callee-save register is used. Assume it is r, at the entry of f, r is pushed, at the exit, it is popped. If r is a caller-save register, it has to be pushed before the invocations to g() and h() and popped after the calls.

   c. A caller-save register is used. Because the variable is not live, it is not needed to save a copy of r before the calls to f(y) and f(2). If r is callee-save, it has to be pushed at the entry and popped at the exit of h( );

4. (10 pts) Given the following program

```
int f (int i) {

    int j,k;
    j=i*i;
    k=i? f(i-1):0;
    return k+j;

}

void main ( ) {  f(100000); }
```

   a. Imagine a compiler that passes parameters in registers, wastes no space providing backup storage for parameters in registers, does not put local variables in registers, and in general makes stack frames as small as possible. How big should each stack frame for *f* be, in words? Please explain the reason.

      The stack frame has 1 word for j, 1 word for k, 1 for the parameter of the recursive call, 1 for return address, 1 for the pointer to the old stack frame, 1 for the flags register. (The last one is optional).

   b. What is the maximum memory use of this program, with such a compiler?

      100000*6+ 2 as the last frame has only the locals but not the parameter, etc. as the call does not occur.
      Partial credit will be given to (100000+1)* 6 or 100000*6.

5. (10 pts) Given the following program

```
void foo(char * s) {
  int i;
  char c[4];
  i=0;
  for (i=0;i<strlen(s);i++) c[i]=s[i];
  printf("i=%x\n",i);
}

int main () {
  foo ("aaaaaaaa");
}
```

Assuming the compiler places variables next to each other in their declaration order in the active record, i.e. *i* is at a higher address than *c*. What is the output of this program? Why? Note that we also assume the compiler does not deploy stack-smash protection.

62. Overflow followed by an increment. 61 will receive 75% credit.