CS 480 Final Review

What are the pieces of the compiler tool chain? What are the jobs of each?

List the layers of the compiler stack from input source code to emitted machine code.
Why is the parser often separate from the code generator?

How is machine code different than intermediate code?

The ternary operator in C (test ? val1 : val2) means if test is true use val1 as the value of the expression, otherwise use val2 as the value of the expression.

Assume we added the ternary operator to C-lite. Produce the complete set of opcodes for the main() function below (and only the main() function). Assume foo() starts at file offset 0 and contains no constants. Assume that main() starts at file offset 100.

```
int x;
int y;

int foo(x, y)
int x, y;
{ /* do stuff */ }

main()
{
  int x;

  input (&x, &y);
  foo( x > y ? 9 : x + y , x || (y >9));
}
```

The expression **`(*x)++`** is not allowed by our BU grammar. However, the precedence table does not produce an error when this is parsed. Why is that? Use the following snippet of the grammar to answer the question. Be as complete as possible with your answer. How would you need to change the grammar to have the above expression cause an error in the precedence table (or is it impossible?)? How would that change complicate other parts of the compiler?

```
unexp -> lvalue autoop  | & lvalue | * unexp | negop  unexp | primary
primary -> ( exp )  | lvalue | constant | func
lvalue -> var | (  lvalue )
```

Build a CFG to produce strings where all the 1s occur before all the 0s and there are more 1s than 0s. Σ = {2,1,0}

Explain what a shift/reduce conflict is. Provide a grammar and a string that causes a shift/reduce conflict.

```
E -> T * E
T -> T + F
F -> 0 | ... | 9
```

What is the right most derivation of 9 + 0 + 1 * 7 + 1
Left most?
Show the BU parse. What would the operator precedence table look like for this grammar?

What is the associativity of + and * ? The precedence between them?
Can the above grammar be used in a TD parser?
Can you transform the above grammar into one that is usable with a TD parser?
Does this new grammar preserve the associativity of + and * ?

Turn the following into three address code and then build the control flow graph.

```
int k, i;
k=0;
i=0;
input(&x, &y);
for( i=0; i< 100; i++)
{
  k=0;
  for( k = 100;k > 0;k—)
  {
    x = x + y;
    if(x % 2 == 1)
    {
      output(k, x, y--);
    }
  }
  output(i, x, y);
}
```

Perform any optimizations within the basic blocks produced above that you can.

What is a handle with respect to function arguments?

What is a handle with respect to parsing?

Given the following makefile and directory information, list each command that is run if the user types **`make sa3Test`** at the command line. List all assumptions you make.

```
#begin makefile

CC = ./pcc
CFLAGS = -sa3srcdump -sa2tabdump -sa1symdump -sa3quaddump

all: sa3Test

sa3Test: test1.q test1.q.solution.out
     /updates/inter test1.q > test1.q.out
     diff -Bw test1.q.out test1.q.solution.out

test1.q: test1.c
     $(CC) $(CFLAGS) -o test1.q test1.c

test1.c: /var/www/html/chadd/cs480s09/test1.c
     cp /var/www/html/chadd/cs480s09/test1.c test1.c

test1.q.solution.out: test1.c
     /update/pcc $(CFLAGS) -o test1.q.solution test1.c
     /updates/inter test1.q.solution > test1.q.solution.out

#end makefile
```

```
chadd@chaddmachine:~/test/final> ls
total 92
drwxr-xr-x 2 chadd users  4096 2009-05-11 13:14 .
drwxr-xr-x 4 chadd users  4096 2009-05-11 13:10 ..
-rw-r--r-- 1 chadd users   523 2009-05-11 13:13 Makefile
-rwxr-xr-x 1 chadd users 69917 2009-05-11 13:11 pcc
-rw-r--r-- 1 chadd users    44 2009-05-11 13:14 test1.c
```

```
Decode:
 1  27   0   0   0   0   0   0
 2  22   0  10   0   0   0   0
 2  26   2   1   0   0   3   1
 2  26   2   2   0   0   3   2
 2   1   4   1   4   2   3   3
 2  10   4   3   2   3   0   8
 2  26   0   0   0   0   3   4
 2  19   0   0   0   0   0   9
 2  26   0   1   0   0   3   4
 2   2   4   1   4   2   3   5
 2  15   4   5   2   1   0  13
 2  26   0   0   0   0   3   6
 2  19   0   0   0   0   0  14
 2  26   0   1   0   0   3   6
 2  17   4   4   4   6   0  17
 2  26   0   0   0   0   3   7
 2  19   0   0   0   0   0  18
 2  26   0   1   0   0   3   7
 2  18   4   7   0   0   0  31
 2  19   0   0   0   0   0  23
 2  26   4   1   0   0   3   8
 2   7   0   0   0   0   3   1
 2  19   0   0   0   0   0   4
 2   1   4   1   4   2   3   9
 2  20   4   9   0   0   0   0
 2  20   4   2   0   0   0   0
 2  20   4   1   0   0   0   0
 2  25   0   3   0   0   0   0
 2  26   4   2   0   0   3  10
 2   7   0   0   0   0   3   2
 2  19   0   0   0   0   0  20
 2  26   0   0   0   0   1   0
 2  23   0   0   0   0   0   0
 1  21   0   0   0   0   0   1
 1  28   0   0   0   0   0   0
4
0
0
3
10
```