Name: ______________________

Math 211 Final
May 11, 2000

Professor L. Taylor

1. What value is in the Perl variable `$xx` after the statement

```
$xx=&mm(0.2);
```

where the code for `mm` is

```
sub mm{
  $y=shift(@_);
  return($y**2);
  }
```

2. Write the function `mm` from the previous problem in C assuming that the declaration `float mm(float xx);` has already been given.

Name: ____________________

3. Write a program in C that prints to the screen a table of the first two hundred prime numbers. Each line should have the number of the prime, printed with 3 characters followed by a space and then the prime printed with 4 characters and finally a newline. The first two lines should look like

```
  1    2
  2    3
```

Hint: For once, recursion isn't necessary.

Name: ____________________

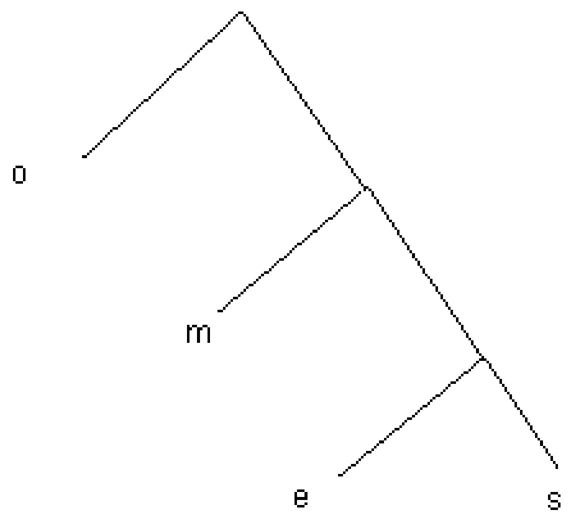
4. A use for binary trees that was mentioned in class but not pursued is for file compression. The scheme is called *Huffman encoding* and works like this. You are given a binary tree like the one below and a string of binary digits (0's and 1's). Each node of the tree with no children has a character associated with it.

To use the tree and the string to decode the message, proceed as follows. Start at the root and look at the first digit. If it is a 0 take the left branch, if it is a 1 take the right branch. Look at the next digit and proceed similarly. Eventually, you will arrive at a node of the tree with no children. Record the character at this node and return to the root of the tree and start the procedure again, beginning with the next digit in the binary string.

1. Given the tree below and the binary string

   `1000111110`

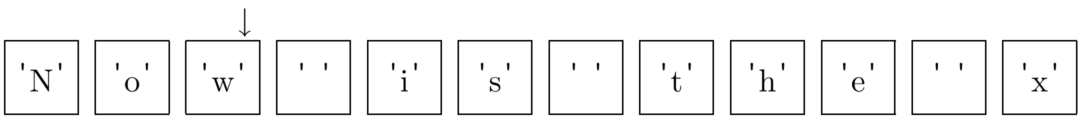
   what familiar 5 letter word do you get?



2. What is the binary string that encodes the letter `s`?
3. The encoding given here for our entire word takes 10 bits. A single character encoded in ASCI takes 8 bits. How many bits does the usual ASCI encoding of this word take?

Name: ______________________________

5. Suppose `cc` is a `char *` (a character pointer) which points to the `'w'`in an array of character boxes which are filled as indicated below. Assume that `x` has been declared as an `int`. Finally, assume that a `short` takes the same space as 2 `char`'s.



1. After the following code is executed, to what character does `cc` point?
```
x=3; x*=x;cc+=x; cc-=2;
```

2. If, instead of the code in 1, the following code is executed, to what character does `cc` point?
```
x=4; cc=(char*)((short*)cc+x);cc-=3;
```

Name: ______________________________

6. What does the following program print on the screen? Be careful.

```
#include <stdio.h>

main(){
  int    a;

a=3;
if( a=2 ) {printf("Hello?\n"); }
printf("%d\n",a);
}
```

****

Answer 1. In Perl `mm` is a function which takes a value and returns its square. The `**` in Perl is exponentiation. Hence `&mm(0.2)` returns $0.2^2 = 0.04$.

Answer 2. This question just asks for the C code for a function which takes one float as input and returns its square. A minimalist solution is

```
float mm(float xx) {
return(xx*xx);
}
```

You can not write `xx**2;` in C - there is no such operation. You can use the math function exponentiation although no one did.

Answer 3. The easiest way I know to do the problem is to declare an array of int's of length 2000 and fill it with the primes. If I had a function which could test for primes, this could be done by

```
index=0;test=2;
while(index<MAXPRIME){
  if( test is prime ) p[index++]=test;
  test++;
  }
```

Then I could print out the answers with a `for` loop. So how do I test for a prime? If I had a list of primes smaller than my number, I could tell if a number was prime by just checking if these smaller primes divided it. But I am constructing such a list so I can write a function which takes my list thus far and my number and does the check. The following code works, although it could be much more efficient.

```
#include <stdio.h>

#define MAXPRIME 200

char primetest(int test, int index, int p[]);

main(){
  int p[MAXPRIME];
  int index, test;

index=0;test=2;
while(index<MAXPRIME){
  if(primetest(test,index,p)==1) p[index++]=test;
  test++;
  }
for(index=0;index<MAXPRIME;index++){
  printf("%3d %4d\n",index+1,p[index]);
  }
}
```

Name: ______________________

```
char primetest(int test, int index, int p[]) {
int ii;

for( ii=0;ii<index; ii++){
  if( test%p[ii]==0) return -1;
  }
return 1;
}
```

Another popular answer is less efficient but does the job.

```
#include <stdio.h>

#define MAXPRIME 200

char primetest(int test);

main(){
  int p[MAXPRIME];
  int index, test;

index=0;test=2;
while(index<MAXPRIME){
  if(primetest(test)==1) p[index++]=test;
  test++;
  }
for(index=0;index<MAXPRIME;index++){
  printf("%3d %4d\n",index+1,p[index]);
  }
}

char primetest(int test) {
int ii;

for( ii=0;ii<test; ii++){
  if( test%ii==0) return -1;
  }
return 1;
}
```

One can of course incorporate the `primetest` function into the main part of the program but this is anti-C. A final choice was to write a function `nextprime`. One version goes like

```
#include <stdio.h>

#define MAXPRIME 200

int nextprime(int test);
```

Name: ______________________

```
main(){
  int index, test;

index=1;test=2;
while(index<=MAXPRIME){
  printf("%3d %4d\n",index++,test);
  test=nextprime(test);
  }
}

int nextprime(int test) {
/* returns next prime bigger than test */
int ii;

test++;
while( 0==0) {/* infinite loop*/
  for(ii=2;ii<test;ii++) {
    if(test%ii==0) {test++;break; }
    }
  if(ii==test) return test;
  }
}
```

Answer 4.

1. The word is moose: 10 gets you to m; 0 gets you to o; 0 again gets another o; 111 gets you to s; and the final 110 gets you to e.
2. As indicated in the answer to 1, s is encoded by 111.
3. Since each character takes 8 bits and there are 5 characters, the usual ASCI encoding takes 8*5=40 bits. Some people got 48 by declaring moose to be a C string and hence ending in a 0. I let this slide although I got the exercise out of a Mathematica book and I have no idea how it handles strings. Finally, there is a nice application of binary trees to the enumeration of the rationals discussed in the current issue of the American Mathematical Monthly if you're interested.

Answer 5. First lets get straight what things are: `cc` is a pointer and `xx` is an int.

1. The first statement sets `xx` to 3; the second says take the value of `xx` and multiply it by `xx`, so `xx` now has value 3*3=9 (the * in the second statement has nothing to do with pointers!); the third statement increments `cc` by 9 and since `cc` started out pointing to `'w'` it now points to `'x'`; the last statement decrements the pointer by 2 so `cc` now points to `'e'`.
2. The first statement sets `xx` to 4. The second is somewhat convoluted because of the type casts. Working from the inside out, first we cast `cc` to be a `short` pointer. Then we add 4 to it. Since a `short` is two `char`'s, `cc` now points to the `' '` after the `'e'`. Now we cast `cc` back to a `char` pointer. Finally the third statement decrements the character pointer by 3 so it now points to `'t'`.

Answer 6. As it says, be careful. The second line of actual code is unusual. The `if( a=2)` does two things. First, it puts a 2 into the `a` variable. This statement has the value

2 (every statement in C has a value) and C evaluates 2 as true so we do the `printf` statement. Hence the answer is

```
Hello?
2
```