CS16 Exam #2 8/14/2012
75 Minutes 100 Points total

Name: _____

1. (20 pts) For this question, assume that integers never overflow and are of infinite size. For each loop, circle if it is an infinite or a finite loop. Also show what the output of running the code is. If there is an infinite amount of output, just show the first three iterations. If there is no output, write "NO OUTPUT".

Code	Finite / Infinite	Output

`int x;`
`for(x=0;x<4;x++)`
`  printf("%i",x);`
	finite
	
	infinite
		0123

`int x;`
`for(x=1;x<1;x++)`
`  printf("%i",x);`
	finite
	
	infinite
		NO OUTPUT

`int x;`
`for(x=4;x>0;x--)`
`  printf("%i",x);`
	finite
	
	infinite
		4321

`int x;`
`for(x=0;x<5;x+=2)`
`  printf("%i",x);`
	finite
	
	infinite
		024

`int x;`
`for(x=0;x<4;x--)`
`  printf("%i",x);`
	finite
	
	infinite
		0-1-2

2. (2 pts) What is wrong with the following code? Fix it so that it will compile without warnings or errors.

void bar(); // lacking a function prototype

```
void foo() {
  printf( "foo" );
  bar();
}

void bar() { printf( "bar" ); }
```

3. (9 pts) For each of the following loops, record the final values of the variables x, y, and z.

Code	x	y	z

`int x = 0, y = 0, z = 0;`
`while( x <= y + z ) {`
`  y = y + 2;`
`  x = y;`
`  y = z;`
`  z++;`
`}`
	2	0	1

`int x = 0;`
`while ( x == 5 ) {`
`  x++;`
`}`
	0	N/A	N/A

`int x = 0;`
`do {`
`  x++;`
`} while ( x == 1 );`
	2	N/A	N/A

`int x = 0;`
`while ( x < 10 ) {`
`  x += 8;`
`  x++;`
`}`
	18	N/A	N/A

`int x;` `for( x = 0; x < 5; x++ ) {}`	5	N/A	N/A

`int x;`
`for( x = 0; x < 10; x++ ) {`
`  if ( x % 2 == 0 ) {`
`  continue;`
`  }`
`  x = x + 5;`
`}`
	13	N/A	N/A

`int x;`
`for( x = 0; x < 10; x++ ) {`
`  if ( x % 2 == 0 ) {`
`  break;`
`  }`
`  x = x + 5;`
`}`
	0	N/A	N/A

4. (2 pts) Declare an array containing 20 `double`s named `arr`. Do not initialize it.

```
double arr[ 20 ];
```

5. (3 pts) In one statement, declare an array named `arr2` that holds 20 `double`s and whose first four elements are `4.4`, `2.5`, `3.3`, and `6.7`, respectively.

```
double arr2[ 20 ] = { 4.4, 2.5, 3.3, 6.7 };
```

6. (3 pts) In one statement, declare an array named `arr3` holding 20 `int`s which are are all initialized to `0`.

```
int arr3[ 20 ] = { 0 };
```

7. (5 pts) Assume that `arr3` is an array holding the integers 0-9. (i.e. `arr3` holds `int`s, and the first element is 0, the second 1, and the last element 9.) With this definition in mind, list the value of each of the following expressions. Write down ERROR if it would fail to compile, or UNDEF if the value is undefined.

Expression	Value
`arr[0]`	0
`arr[1]`	1
`arr[9]`	9
`arr[10]`	UNDEF
`arr[-1]`	UNDEF

8. (15 pts) Consider the following code:

```
#define LENGTH /* some positive integer */
int isEven( int x ) { return ( ( x % 2 ) == 0 ); }
int array[] = { /* some positive integers.  The number of
                 * positive integers == LENGTH */ };
```

Write a the definition for a function named `firstEven` that will return the first even integer in `array`. Return `-1` if `array` contains no even numbers. For full credit, you must call the `isEven` function.

```
int firstEven() {
  int x;
  for ( x = 0; x < LENGTH; x++ ) {
    if ( isEven( array[ x ] ) )
      return array[ x ];
  }
  return -1;
}
```

9. (2 pts) What is wrong with the following code? (Assume that `filename` holds the name of a file that is guaranteed to be able to be opened.)

```
int doSomething( char* filename ) {
  FILE* file = fopen( filename, "r" );
  int retval = getc( file );
  if ( retval != EOF ) {
    fclose( file );
    printf( "Character found: %c\n", (char)retval );
  }

  return retval;
}
```

The file will not always be closed, namely when the file is empty.

10.(15 pts) Write some code that will write all even integers between 2 and 100 (inclusive) to a file named "numbers.txt". Do NOT assume that "numbers.txt" can be opened. Make sure to close the file when you're done with it. Additionally, for full credit, you may not use the modulus (%) operator.

```
FILE* file = fopen( "numbers.txt", "w" );
if ( file != NULL ) {
  int x;
  for( x = 2; x <= 100; x = x + 2 ) {
    fprintf( file, "%i\n", x );
  }
  fclose( file );
}
```

11.(2 pts) For project #2, Bob spends hours testing his code, using hundreds of different inputs for the various prompts. All the tests produce the expected result, and Bob declares that his code is free of bugs. Is this a fair statement? Why or why not?

No. Testing can only find existing bugs. It cannot show that no bugs exist (perhaps you just need another test?).

12.(8 pts) What does the following code print, starting execution at `main`?

```
int x = 0;
void one( int x ) {
  printf( "one: %i\n", x );
}

void two( int notX ) {
  x = 2;
  printf( "two: %i\n", x );
}

void three( int notX ) {
  x = x + 3;
  printf( "three: %i\n", x );
}

void four( int x ) {
  if ( 1 ) {
    int x = 4;
  }
  printf( "four: %i\n", x );
}

void five( int x ) {
  if ( 1 ) {
    x = x + 5;
  }
  printf( "five: %i\n", x );
}

int main() {
  one( x );
  if ( 1 ) {
    int x = 8;
    two( x );
    printf( "main1: %i\n", x );
    three( x );
    printf( "main2: %i\n", x );
  }
  four( x );
  five( x );
  printf( "main3: %i\n", x );
  return 0;
}
```

--OUTPUT--

one: 0
two: 2
main1: 8
three: 5
main2: 8
four: 5
five: 10
main3: 5

13.(14 pts) Consider the following code:

```
struct Foo {
  int i;
  int* ip;
};

int i = 5;
int* ip = &i;
struct Foo foo = { i, ip };
struct Foo* foop = &foo;
```

List the type of each of the following expressions. If the expression is invalid, write ERROR. The first one has been done for you.

Expression	Type
i	int
*i	ERROR
&i	int*
*ip	int
&ip	int**
foo.i	int
foo.ip	int*
foo->i	ERROR
foo->ip	ERROR
foop.i	ERROR
foop.ip	ERROR
foop->i	int
foop->ip	int*
(*foop).ip	int*
foo.moo	ERROR