CSC112 Fall 2010
Fundamentals of Computer Science
Practice Problems

1. What is the 8-bit base 2 representation of the (base 10) number 37 ?

   Answer: `0 0 1 0 0 1 0 1`

2. Using two's complement representation, what is the 8-bit representation of the base 10 number -37 ?

   Answer: `1 1 0 1 1 0 1 1`

3. What would the following program print ?

```
#include <iostream>
using namespace std ;
int main()
{
    cout << (25 & -9) << endl ;
}
```

   Answer: `17`

4. The program in problem 3 works, but the one given below does not compile. I.e., g++ prints an error message. Why ?

```
#include <iostream>
using namespace std ;
int main()
{
    cout << 25 & -9 << endl ;
}
```

   Hint: If I name the program file "u.cc", then the error message is:

```
u.cc: In function 'int main()' :
u.cc:5: error: invalid operands of types 'int' and '<unresolved overloaded
function type>' to binary 'operator<<'
```

   Answer: The error here is that the operator precedence does not cause the bit-wise "and" to be performed first. The stream operator is higher precedence, so the compiler "tries" to translate the sub-expression `-9 << endl` . But, the left argument to the stream operator is an integer, and this operator requires a stream object (e.g., cout) as the left argument.

5. *Note: This one is an unlikely test question, but try it on your computer.* The following program is intended to print the alphabet (lower case). Does this work ?

```
#include <iostream>
using namespace std ;
int main()
{
   char letter ;
   for ( letter = 'a' ; letter <= 'z' ; letter++ ) cout << letter ;
   cout << endl ;
}
```

Answer: Yes. Output is:

```
abcdefghijklmnopqrstuvwxyz
```

6. Write a function to accept an array of integers `a`, and the number of integers `n` stored in the array. Your function should return the largest integer in the array. The function header should be:

```
int find_largest( int a[], int n )
```

Answer: See code segment below.

```
int find_largest( int a[], int n )
{
   int i ;
   int largest ;

   largest = a[0] ;   //  Use the first element in the array as our
                      //  candidate for the largest element.

   //  The following loop goes through the rest of the array.
   //  If our candidate 'largest' is smaller than a[i], it can
   //  not be the largest, Right ?   In that case, we replace the
   //  value of 'largest' with a better candidate, i.e.,  a[i]
   //  Once the entire array is processed, we know that 'largest'
   //  contains the largest element in the array.

   for ( i = 1 ; i < n ; i++ ) {
      if ( largest < a[i] ) largest = a[i] ;
   }

   return(largest) ;   //  The return statement passes the value
}                      //  back to the caller.
```

7. Give an expression that maps a 2-D index (i, j) to a 1-D index when storing a dynamically allocated matrix in row-major order.

   Answer: Let m and n be the number of rows and number of columns in the matrix respectively. Suppose the a is the matrix declared and allocated as:

   ```
   int * a ;
   a = new int [ m*n ] ;
   ```

   Then the correct expression (using row-major storage order) for accessing row `i` and column `j` is:

   ```
   a[ i * n + j ]
   ```

8. Write a program that reads an integer `n`, and dynamically allocates an array of `n` integers. *Note: this would normally be part of a larger project, but for purposes of the question, just show how to declare and allocate the array.*

   Answer: See the program below.

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main()
{
    //  Declarations needed for a dynamically allocated array.
    int n ;
    int * a ;

    //  Prompt the user
    cout << "Enter array size : " << endl ;
    //  Read the input
    cin >> n ;

    a = new int[n*sizeof(int)] ;   //  Allocate memory.


   //  The rest of the application would normally follow.


   //  When memory is no longer needed, release our claim to it.
   delete [] a ;
}
```

9. Explain the difference between *pass by value* and *pass by reference* when passing parameters to a C++ function.

   Answer: Pass by value copies the value of the actual parameter (in the caller) to the formal parameter (in the called function). Changes to the parameter in the called function does not change the value of the actual parameter in the caller.

   Pass by reference causes changes to the formal parameter (in the function) to be reflected in the actual parameter (in the caller). There are two ways to do this.

   We can use C++ pointers to create pass-by-reference behavior. In reality, when we pass the pointer, it is copied from the (pointer) actual parameter, to the (pointer) formal parameter. The function uses pointer reference to store into the memory location that is indicated by the pointer. This memory location is also the memory location of the corresponding variable in the caller.

   We can also use C++ references to create pass-by-reference behavior. Using this language feature simplifies the syntax and partially automates the process of passing pointers described above.

   See the handout "ex8" distributed in class for examples of the various parameter passing methods. The handout is available on-line at:

   `http://menehune.opt.wfu.edu/CSC112/ex8.ps`

   Also, the source code can be downloaded from:

   `http://menehune.opt.wfu.edu/CSC112/CC/ex8.cc`

10. Explain the difference between automatic storage and static storage in a function. For example, what's the difference between the variables `a` and `b` ? *Hint: The answer is NOT that "a is one and b is two".*

```
int example(int u )
{
    int a = 1 ;
    static int b = 2 ;
         .
         .
         .
}
```

   Answer: Memory for the variable `a` is allocated within the activation record for function `example` on the system stack. This means that the memory is allocated (i.e., activation record is created) at the time the function is called. When the function returns, this memory is no longer available for use. Subsequent calls to other functions will re-occupy (and over-write) the memory formerly used by the variable `a`. The initialization code `a=1` is performed every time the function is called.

   Memory for the variable `b` is allocated in a special "static" memory segment. This memory segment is persistent throughout the run of the program. The initialization code `b=2` is only performed once; it is done at the time the program is loaded into memory by the operating system. Changes to the variable `b` made by the function will be retained from one call to the function to the next.

See the example on-line:

```
http://menehune.opt.wfu.edu/CSC112/ex7.ps
```

Source code for this example is also available on-line:

```
http://menehune.opt.wfu.edu/CSC112/CC/ex7.cc
```

11. What would this program print ?

```
#include <iostream>
using namespace std ;

int main()
{
    int a ;
    int * p ;

    a = 7  ; p = &a ; a += a ;
    cout << *p << endl ;
}
```

Answer: The program prints `14`.

12. This last one is too difficult for a test question, but we will discuss it in class.

```
#include <iostream>
using namespace std ;

int main()
{
    int a, b ;
    int * p ;

    a = 7  ; b = 11 ; p = &a ;
    cout << *(p-1) << endl ;
}
```

Answer: The program prints `11`.