1. During the class we have explained that structures in C/C++ (or records in Pascal) are aligned in the memory. Please use the rules for a 64-bit x86 processors to align the following structure. You can refer to http://en.wikipedia.org/wiki/Data_structure_alignment for more information.

   (a) (10 points) Please draw the allocation of each variable in the memory and compute how much memory is used per structure.

   ```c
   struct MyMixedData {
      long long a;
      float b;
      char c;
      int d;
      short e;
      char f;
      short g;
   };
   ```

   (b) (5 points) Without using the composite data types, how much memory is required to store the above data?

   (c) (5 points) Can you rewrite the above structure to conserve memory? (As what advanced compilers do in optimization)

2. (15 points) Explain the meaning of the following C declarations:

   ```c
   double *a[n];
   double (*b)[n];
   double (*c[n])();
   double (*d())[n];
   ```

3. (10 points) In the following Pascal code, which of the variables will a compiler consider to have compatible types under structural equivalence? Under strict name equivalence? Under loose name equivalence?

   ```pascal
   type T = array [1..10] of integer
   S = T
   A : T
   B : T
   C : S
   D : array [1..10] of integer
   ```

4. (10 points) Consider the following C declaration, compiled on a 32-bit Pentium machine:
If the address of A[0][0] is 1000 (decimal), what is the address of A[3][7]? and A[5][9]? (Remember the address/memory alignment is at 32 bit)

5. Tail recursions can improve the performance of programs, since it can be easily transformed into a non-recursive version automatically by a compiler.

(a) (10 points) Write a tail-recursive function in any programming language to compute n factorial

\[ n! = \prod_{i=1}^{n} i = 1 \times 2 \times \cdots \times n \]

(b) (5 points) Is it possible to write a tail-recursive version of the classic quicksort algorithm? (You may want to review Cormen’s Algorithm textbook)

6. (10 points) Applicative and normal evaluation order. Consider the following C code. In what case will the macro call be faster than the function call and vice versa?

   ```c
   #define select_m(n,a,b) ((n) ? (a) : (b))
   inline double select_f (int n, double a, double b) { return n ? a : b; }
   #define abs_m(x) ((x) >= 0 ? (x) : -(x))
   inline double abs_f (double x) { return x >= 0 ? x : -x; }
   ```

7. (10 points) Postscript is a programming language usually used to create vector graphics. It is dynamically typed and concatenative programming language. Please briefly explain what kind of language Postscript is and why operator associativity and precedence is not a problem in Postscript. Please do a little research on why these types of programming languages are usually faster.

8. (10 points) Some compilers are very smart (perhaps too smart) and can rearrange mathematical expressions, for example, let a, b, c, d all be integers:

   a = b / d + c / d;

   into

   a = (b + c) / d;

   which eliminates one division. Is this a safe or unsafe action? What potential problems can arise from this action?