COMP 356 Homework Assignment 2

Acknowledgment. Question 1 was written by Prof. Tim Wahls.

Instructions: Turn in a hard copy of question 1. Submit your solution to question 2 to Moodle.

Question 1 (10 pts)

Consider the following BNF definition of statements in an imaginary (and tiny) programming language:

⟨stmt⟩ → ⟨assign-stmt⟩ ;
⟨assign-stmt⟩ → ⟨var⟩ = ⟨expr⟩
⟨var⟩ → `d` | `d1` | `f` | `i`
⟨expr⟩ → ⟨var⟩ | ⟨expr⟩ + ⟨expr⟩ | ⟨expr⟩ ∗ ⟨expr⟩ | ⟨assign-stmt⟩

(a) Draw a parse tree for the statement:

```
d1 = d = f + i;
```

(b) Annotate each nonterminal in the parse tree with its actual type using the typechecking rules of Java. Here, *actual type* has the same meaning as example 3.6 in the textbook, and as discussed in class. Assume the following intrinsic type attributes have already been assigned: variables `d` and `d1` are of type double, `f` is of type float, and `i` is of type int.

(c) Are there any nodes where the *expected type* (again, as defined in class and example 3.6 in the textbook) differs from the actual type? If so, would any compilation errors result under the Java compiler rules?

Question 2 (30 pts)

Important note, added 9/15/14: Your program must compile and run without errors on the Tome macs. Please test it there before submitting.

Write a C program that employs, and demonstrates your understanding of, all the following features:

- `#define`
- enums
- pointers and pointer arithmetic
- addresses
- unions
- sizeof
- employs at least two commandline arguments as integers
- a function that creates and returns a 1D array
- passing a 1D array as a parameter to a function
- a function that returns two separate integers as out parameters
- 2D arrays

Your program doesn't have to be useful, or even particularly meaningful. Feel free to be creative and come up with your own examples of the above for any or all parts of this question. Alternatively, you can fill in the missing parts of the following framework (available from the course webpages as `c-example-skeleton.c`):

```
// include required header files
...

// define your graduation year using a #define
...

// define an enum consisting of the five weekdays: MONDAY to FRIDAY
...

// define a Boolean function that is true if the parameter
// is the last day of the workweek (i.e. Friday)
... isEndOfWeek(...) {
  ...
}

// declare a union type that can contain an int or a double
...

// declare a struct type that contains an int and a double
...

// return a pointer to a newly-created array,
// containing the first arraySize even numbers
... createArrayOfEvens(int arraySize) {
  ...
}

// return the sum of the elements in a given array of ints, which has the given
// length
... addArrayElements(... array, ... length) {
  ...
}

// this function takes an array of ints as input,
// and returns the first two values in the array as outputs;
// both values should be returned as out-parameters
void getFirstTwoArrayElements(...) {
  ...
}

// Main function for the program
int main(int argn, char* argv[]) {
  // print out your graduation year using the #define
  printf("My graduation year is %d\n", ...);

  // use the enum you defined earlier
  if (isEndOfWeek(WEDNESDAY)) {
    printf("Wednesday is the last day of the week\n");
```

```
} else {
  printf("Wednesday is not the last day of the week\n");
}

// check that the user specified enough command line arguments, and
// exit with an error message if not
...

// convert the first two commandline arguments to integers, and
// store them in a and b respectively, then print out their sum
int a = ...
int b = ...
printf("The sum of the first two commandline arguments is %d\n", a+b);

// declare, initialize and print the value of xVal
int xVal = 5;
printf("The value of variable xVal is %d\n", xVal);

// declare and initialize a variable storing the address of xVal,
// then print out the resulting address in hexadecimal
...
printf("The address of variable xVal in hexadecimal is ...", ...);

// change the value of xVal using only its address
// (i.e. without referring to xVal in the program)
...
printf("The value of variable xVal is now %d\n", xVal);

// Declare a variable of your union type
...

// give the variable some integer value, and print it out
...
printf("The value of the union when storing an integer is ...", ...);

// give the variable some double value, and print it out
...
printf("The value of the union when storing a double is ...", ...);

// print sizes of int, double, and your union
printf("The size of an int on this machine is %d bytes\n", ...);
printf("The size of a double on this machine is %d bytes\n", ...);
printf("The size of the union on this machine is %d bytes\n", ...);

// declare a variable of your struct type, and initialize all members to some values
...

// print out the values stored in the struct, and the size of the struct
```

```
  printf("The value of the int in the struct is ...", ...);
  printf("The value of the double in the struct is ...", ...);
  printf("The size of the struct is %d bytes\n", ...);

  // create an array containing the first 20 even the numbers,
  // by calling the function createArrayOfEvens defined earlier
  ...

  // print out the value of element 15, using array notation
  printf("The 15th even number (computed via array notation) is %d\n", ...);

  // print out the same value again, but this time computed using pointer arithmetic
  // i.e. no square brackets are allowed
  printf("The 15th even number (computed via pointer arithmetic) is %d\n", ...);

  // print out the sum of the first five even numbers, by calling
  // the function addArrayElements defined earlier
  printf("The sum the first 5 even numbers is %d\n", ...);

  // declare two ints that will be used to store the first two even numbers
  int val1, val2;

  // store the first two even numbers in val1, val2 respectively,
  // using a single call to the function getFirstTwoArrayElements
  // i.e. no assignment statements are permitted
  ...

  // print out the values you obtained to check that it worked
  printf("The first two even numbers are %d and %d\n", val1, val2);

  // declare a 10x10 array of ints called "products",
  // and initialize it so that products[i][j] contains the
  // product of i and j
  ...

  // print out the product of 3 and 7 by looking up the value in the array
  // i.e. no multiplication is permitted
  printf("The product of 3 and 7 (computed by array lookup, NOT multiplication) is %d\n", .

  // free any memory that was allocated in this program
  ...
}
```