Week 3 Problem Set

Dynamic Data Structures

[Show with no answers] [Show with all answers]

1. (Memory)

   Given the following definition:

   ```
   int data[12] = {5, 3, 6, 2, 7, 4, 9, 1, 8};
   ```

   and assuming that *&data[0] == 0x10000*, what are the values of the following expressions?

data + 4
*data + 4
*(data + 4)
data[4]
*(data + *(data + 3))
data[data[2]]

   [show answer]

2. (Pointers)

   Consider the following piece of code:

   ```
   typedef struct {
      int   studentID;
      int   age;
      char  gender;
      float WAM;
   } PersonT;

   PersonT per1;
   PersonT per2;
   PersonT *ptr;

   ptr = &per1;
   per1.studentID = 3141592;
   ptr->gender = 'M';
   ptr = &per2;
   ptr->studentID = 2718281;
   ptr->gender = 'F';
   per1.age = 25;
   per2.age = 24;
   ptr = &per1;
   per2.WAM = 86.0;
   ptr->WAM = 72.625;
   ```

   What are the values of the fields in the *per1* and *per2* record after execution of the above statements?

   [show answer]

3. (Memory management)

   Consider the following function:

   ```
   /* Makes an array of 10 integers and returns a pointer to it */

   int *makeArrayOfInts() {
      int arr[10];
      int i;
      for (i=0; i<10; i++) {
         arr[i] = i;
      }
      return arr;
   }
   ```

   Explain what is wrong with this function. Rewrite the function so that it correctly achieves the intended result using `malloc()`.

   [show answer]

4. (Memory management)

   Consider the following program:

   ```
   #include <stdio.h>
   #include <stdlib.h>
   #include <assert.h>

   void func(int *a) {
      a = malloc(sizeof(int));
      assert(a != NULL);
   }

   int main(void) {
      int *p;
      func(p);
      *p = 6;
      printf("%d\n",*p);
      free(p);
      return 0;
   }
   ```

   Explain what is wrong with this program.

   [show answer]

5. (Dynamic arrays)

   Write a C-program that

   - takes 1 command line argument, a positive integer *n*
   - creates a dynamic array of *n* `unsigned long long int` numbers (8 bytes, only positive numbers)
   - uses the array to compute the *n*'th Fibonacci number.

   For example, **`./fib 60`** should result in 1548008755920.

   Hint: The placeholder `%llu` (instead of `%d`) can be used to print an `unsigned long long int`. Recall that the Fibonacci numbers are defined as Fib(1) = 1, Fib(2) = 1 and Fib(*n*) = Fib(*n*-1)+Fib(*n*-2) for *n*≥3.

   An example of the program executing could be

   ```
   prompt$ ./fib 60
   1548008755920
   ```

   If the commad line argument is missing, then the output to `stderr` should be

   ```
   prompt$ ./fib
   Usage: ./fib number
   ```

   We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to this exercise. It expects to find a program named* `fib.c` *in the current directory. You can use dryrun as follows:*

   ```
   prompt$ 9024 dryrun fib
   ```

   [show answer]

6. **Challenge Exercise**

   Write a C-program that takes 1 command line argument and prints all its *prefixes* in decreasing order of length.

   - You are not permitted to use any library functions other than `printf()`.
   - You are not permitted to use any array other than `argv[]`.

   An example of the program executing could be

   ```
   prompt$ ./prefixes Programming
   Programming
   Programmin
   Programmi
   Programm
   Program
   Progra
   Progr
   Prog
   Pro
   Pr
   P
   ```

   [show answer]