15-122 : Principles of Imperative Computation, Fall 2015

Written Homework 11

Due: **Friday November 20, 2015**

Name: _____

Andrew ID: _____

Section: _____

This written homework will deal with some introductory C concepts.

Print out this PDF double-sided, staple pages in order,
and write your answers on these pages *neatly*.
The assignment is due by 6:00pm
on **Friday November 20, 2015**.
You can hand in the assignment to your TA or in the box outside
of GHC 4117. You must hand in your homework yourself;
do not give it to someone else to hand in.

Question	Points	Score
1	2	
2	3	
3	4	
4	6	
Total:	15	

2pts 1. **Contracts in C**

The code below is taken from the lecture notes on hash sets in C0. This is also legal C code (assuming all the right definitions are available), but the contracts will not be checked in C.

```
elem hset_lookup(hset H, elem x)
//@requires is_hset(H);
//@requires x != NULL;
//@ensures \result==NULL || elem_equiv(\result, x);
{
  int i = elemhash(H, x);
  for (chain* p = H->table[i]; p != NULL; p = p->next) {
    //@assert p->data != NULL;
    if (elem_equal(p->data, x)) return p->data;
  }
  return NULL;
}
```

Rewrite the function in the box on the next page as follows:

- Insert assignment statements so that all return statements have the form `return result`. (In other words, use the variable `result`, defined on the next page, to hold the return value for all cases and use this variable in your postcondition.)
- Insert any necessary C contracts so that, when compiled with the flag `-DDEBUG`, contracts will be checked as they would be in C0 with the flag `-d`.

Do *not* simplify any contracts even if it is immediately obvious from the context that you could do so. You may omit the C0 contracts (lines beginning `//@`) even though in practice we might like to keep them.

```
elem hset_lookup(hset H, elem x) {









  elem result;

















}
```

3pts 2. **Allocating and freeing memory in C**

Here is a leaky C program that works with NULL-terminated linked lists. We've omitted the code for `print_list` because it can't leak any memory. Contracts have been omitted for the sake of space.

```
 1 typedef struct list_node list;
 2 struct list_node {
 3   int data;
 4   list* next;
 5 };
 6 void free_list(list* L) {
 7   list* current = L;
 8   while (current != NULL) {
 9     list* next = current->next;
10     free(current);
11     current = next;
12   }
13   return;
14 }
15 void sum(list* L) {
16   list* sum = xmalloc(sizeof(list));
17   sum->data = 0;
18   list* current = L;
19   while (current != NULL) {
20     sum->data += current->data;
21     current = current->next;
22   }
23   L->data = sum->data;
24   L->next = NULL;
25   return;
26 }
27 int main() {
28   list* current = NULL;
29   for (int i=0 ; i<10 ; i++) {
30     ASSERT(0 <= i);
31     list* new = xmalloc(sizeof(list));
32     new->data = i;
33     new->next = current;
34     current = new;
35   }
36   printf("Initial list: "); print_list(current);
37   sum(current);
38   printf("Summed list: "); print_list(current);
39   return 0;
40 }
```

In the table below, give the line number of each line that leaks memory. A line is considered to leak memory if, as a result of executing it, some allocated memory has not been freed, and no further references to that memory are possible. Returning from the `main` function without deallocating everything that was allocated is considered a leak (even though the operating system will clean it up).

Indicate how to fix the leak(s) by writing any extra code that needs to be added, with the line numbers between which it should be inserted.

Line number of leak	Code to insert	Lines to insert between

4pts 3. **Pass by reference and arrays versus pointers in C** The following little program allocates and initializes an array of integers, then calls a function to swap two of its elements. Rewrite the function `main` in the box below to use array notation instead of pointer notation wherever possible.

```
#include <stdlib.h>
#include <stdio.h>
#include "lib/xalloc.h"
#include "lib/contracts.h"

void swap(int *x, int *y) {
  REQUIRES(x != NULL && y != NULL);
  int t = *x;
  *x = *y;
  *y = t;
  return;
}

int main() {
  int* A = xmalloc(sizeof(int) * 10);
  for (int i = 0 ; i < 10 ; i++) {
    ASSERT(0 <= i);
    *(A + i) = i;
  }
  ASSERT(*(A+2) == 2);
  ASSERT(*(A+4) == 4);
  swap(A+2, A+4);
  ASSERT(*(A+2) == 4);
  ASSERT(*(A+4) == 2);

  printf("All tests passed.\n");
  return 0;
}
```

```
int main() {









}
```

4. **C Program Behavior**

Each of the following C programs contains one or more errors. *Briefly* explain what is conceptually wrong with each example. No credit will be given if you simply copy error messages from the compiler, the runtime system, or valgrind. Of course you are encouraged to use these tools to help you understand the problems.

1pt (a)
```
#include <stdio.h>
#define DIV(X,Y) (X/Y)

int main() {
  int c = DIV(10-1, 2+3);
  printf("(10-1)/(2+3) is = %d\n", c);
  return 0;
}
```

Solution:

1pt (b)
```
#include <stdlib.h>
#include "lib/xalloc.h"

int main() {
  int *A = xmalloc(100);
  for (int i=0; i<100; i++)
    *(A+i) = i*i;
  free(A);
  return 0;
}
```

Solution:

1pt (c)
```
#include <stdio.h>
int main() {
  char* s = "1 is the loneliest number";
  printf("s: %s\n", s);
  *s = '0';
  printf("s: %s\n", s);
  return 0;
}
```

Solution:

1pt (d)
```
#include <stdlib.h>
#include "lib/xalloc.h"
#include "lib/contracts.h"

int main() {
  int* A = xmalloc(sizeof(int) * 10);
  for (int i = 1 ; i < 10 ; i++) {
    ASSERT(1 <= i);
    *(A + i) = i;
  }
  free(A+1);
  return 0;
}
```

Solution:

1pt (e)

```
#include <stdlib.h>
#include <stdio.h>
#include "lib/xalloc.h"
#include "lib/contracts.h"

int main() {
  int* A = xmalloc(sizeof(int) * 10);
  printf("Before: %d\n", A[0]);
  for (int i = 0 ; i < 10 ; i++) {
    ASSERT(0 <= i);
    A[i] = i;
  }
  printf("After: %d\n", A[0]);
  free(A);
  return 0;
}
```

Solution:

1pt (f)

```
#include <stdlib.h>
#include <stdio.h>
#include "lib/xalloc.h"
#include "lib/contracts.h"

int main() {
  int* A = xmalloc(sizeof(int) * 10);
  int* B = A+3;
  for (int i = 0 ; i < 10 ; i++) {
    ASSERT(0 <= i);
    A[i] = i;
  }
  free(A);
  printf("B: %d\n", *B);
  return 0;
}
```

Solution: