

# Example Solutions for Programming Languages G22.2110 Summer 2007 hw05

Assigned Th 6/14/2007, due We 6/27/2007 at 1pm.

These are example solutions. Please keep in mind that often, there is not just one correct solution to a question. If you come up with different answers to the homework, then it may be that both your answers and these answers here are correct. Of course, these answers here may also contain mistakes; if you spot some, please let us know so we can correct them.

---

## Reading Assignments

- For lecture on 6/14/2007: Scott 7.7.0-7.7.1;  
Drexel 1-10 ([http://einstein.drexel.edu/courses/CompPhys/General/C\\_basics/](http://einstein.drexel.edu/courses/CompPhys/General/C_basics/))
  - For lecture on 6/28/2007: Scott 7.1.0-7.1.3, 7.2.0-7.2.2;  
Wheeler 1-6, 8-9 (<http://www.adahome.com/Tutorials/Lovelace/master.htm>)
- 

## Homework Assignments

1. Scopes (14 = 10 + 4 points)
  - 1a. (10 points) Write a Scheme function `list-sort` that takes one list argument and returns a version of the list that is sorted in non-descending order. Like in hw04, your function should use helper functions `list-min` and `remove-first`, but this time, they should be declared by a `letrec` nested in `list-sort`.

```
Example solutions
(define (list-sort x)
  (if (null? x) x
      (letrec ((list-min
                 (lambda (y)
                   (let ((h (car y)) (t (cdr y)))
                     (if (null? t) h
                         (min h (list-min t))))))
            (remove-first
             (lambda (n z)
               (if (null? z) z
```

```
(if (= n (car z)) (cdr z)
    (cons (car z) (remove-first n (cdr z))))))
(let ((m (list-min x)))
    (cons m (list-sort (remove-first m x))))))
```

- 1b. (4 points) Could you define the helper functions using `let` or `let*` instead of `letrec`? If yes, how? If no, why not?

Example solutions

No, because they need to call themselves recursively, and with `let` or `let*`, the scope of the defined identifier does not start before its definition, so the identifier is undefined at the call site.

2. Low-level memory management (10 = 3 + 4 + 3 points)

Read the example code from

<http://www.cs.nyu.edu/courses/summer07/G22.2110-001/hw05-c-example.txt>

- 2a. (3 points) There is a bug in this program. What, if anything, is the symptom for the bug when you run the program?

Example solutions

On my platform, the program produces the following error message:  
a.out(1503) malloc: \*\*\* error for object 0x300150: double free

- 2b. (4 points) What is the root cause of the bug? What kind of bug is this?

Example solutions

Even though the bug manifests itself as a double deletion, the root cause is a dangling reference. Function `listDequeue` fails to null out the reference to the head of the list when the list has only one element. Therefore, `listDestroy` later finds a non-null pointer and tries to call `free` on it.

- 2c. (3 points) Show how to fix the bug.

Example solutions

Add the following lines at the end of function `listDequeue()`:  
if (NULL == h)  
list->head = NULL;

3. Pointer arithmetic (26 = 10 + 16 points)

- 3a. (10 points) Consider the following C program:

```
#include <stdio.h>
int main(int argc, char** argv) {
    int n = 0;
    char *vowels = "aeiou", **p, *q, *r;
    for (p = argv + argv - 1; p >= argv; p--)
```

```

    for (q = *p; '\0' != *q; q++)
        for (r = vowels; '\0' != *r; r++)
            if (*q == *r)
                n++;
    printf("the arguments contain %d vowels\n", n);
    return 0;
}

```

Rewrite the program to use array subscripting instead of pointer arithmetic.

*Example solutions*

```

#include <stdio.h>
int main(int argc, char** argv) {
    int n = 0, i, j, k;
    char *vowels = "aeiou";
    for (i = argc - 1; i >= 0; i--)
        for (j = 0; '\0' != argv[i][j]; j++)
            for (k = 0; '\0' != vowels[k]; k++)
                if (argv[i][j] == vowels[k])
                    n++;
    printf("the arguments contain %d vowels\n", n);
    return 0;
}

```

3b. (16 points) Consider the following C program:

```

#include <stdlib.h>
int main(int argc, char** argv) {
    float x[5][5];
    float (*a)[5] = &(x[3]);
    float (*b)[5] = a;
    b--;
    float* c = &(x[3][4]);
    float* d = &(c[-1]);
    unsigned e = x[3] - x[1];
    unsigned f = &(x[3]) - &(x[1]);
    unsigned g = (char*)(x[3]) - (char*)(x[1]);
    float* h = x[3] - 2;
    return 0;
}

```

Assume that this program runs on an architecture where pointers are 4 bytes large, and the sizes of char and float are 1 and 4 bytes, respectively. Further assume that the value of pointer variable x is the address 100. What are the values of variables a thru h just before the return statement?

*Example solutions*

```

a = &(x[3]) = 100+3*sizeof(float[5]) = 100+3*5*4 = 160
b = a-1 = 160-sizeof(float[5]) = 160-5*4 = 140

```

```

c = &(x[3][4]) = &(x+3*sizeof(float[5])+4*sizeof(float)) = 100+3*5*4+4*4
  = 176
d = &(c[-1]) = 176-sizeof(float) = 176-4 = 172
e = x[3]-x[1] = (160-120)/sizeof(float) = 40/4 = 10
f = &(x[3])-&(x[1]) = (160-120)/sizeof(float[5]) = 40/20 = 2
g = (char*)(x[3])-(char*)(x[1]) = (160-120)/sizeof(char) = 40/1 = 40
h = x[3]-2 = 100+3*sizeof(float[5])-2*sizeof(float) = 100+3*5*4-2*4
  = 160-8 = 152

```

4. Ada (0 points)

Start teaching yourself Ada by doing the following:

- 4a. If possible, find peers (other students who want to learn Ada together with you) and gurus (people who already know Ada, whom you can ask questions when you get stuck).
- 4b. Make sure you have access to an Ada95 compiler in your preferred working environment. For example, you can use GNAT, the GNU Ada compiler, which is part of gcc and may therefore already be installed on your platform. Here is online documentation for GNAT: [http://gcc.gnu.org/onlinedocs/gcc-3.4.6/gnat\\_ugn\\_unw](http://gcc.gnu.org/onlinedocs/gcc-3.4.6/gnat_ugn_unw).
- 4c. Read Sections 1-6 and 8-9 of the Ada tutorial (see reading assignment above). Along the way, try things out with the Ada95 compiler that you installed in Step b.
- 4d. Familiarize yourself with the structure of the online Ada 95 reference manual, and find where the predefined language environment is documented.

5. C (0 points)

This question is optional. It is provided to help you improve your C skills if you are not yet experienced with that language.

5a. I/O

Write a program that prompts the user for his or her name, reads the name from input, then politely greets the user by name. You can assume that the user response does not exceed 100 characters. Here is an example interactive session:

```

What is your name?
Bob
Hello, Bob, nice to meet you!

```

*Example solutions*

```

#include <stdio.h>
#include <string.h>
int main(int argc, char** argv) {
    char name[100];
    printf("What is your name?\n");
    fgets(name, 100, stdin);
    printf("Hello, %.*s, nice to meet you!\n", (int)strlen(name)-1, name);
    return 0;
}

```

5b. Libraries

Write a program that uses C library functions to compute  $\sqrt{2}$ ,  $\sin(3.5)$ , and  $e^{2.5}$ , and then prints the results like this (don't worry if the decimals aren't 100% identical):

```
square root of 2.0:    1.414214
sine of 3.5:          -0.350783
e to the power of 2.5: 12.182494
```

*Example solutions*

```
#include <stdio.h>
#include <math.h>
int main(int argc, char** argv) {
    double sqrt2 = sqrt(2);
    double sin35 = sin(3.5);
    double exp25 = exp(2.5);
    printf("square root of 2.0:    %f\n", sqrt2);
    printf("sine of 3.5:          %f\n", sin35);
    printf("e to the power of 2.5: %f\n", exp25);
    return 0;
}
```

5c. Types

The following code creates a variable `c` with the character value 'Z', and then prints a description and the value of the variable:

```
char c = 'Z';
printf("name c, type char, value %c\n", c);
```

Extend this program by creating and printing more variables of different types. Your program should produce the following output:

```
name c, type char, value Z
name f, type float, value 3.141
name i, type int, value 42
name ia, type int array, value { 1, 4, 9, 16 }
name s, type string, value hello
```

*Example solutions*

```
#include <stdio.h>
int main(int argc, char** argv) {
    char c = 'Z';
    float f = 3.141;
    int i = 42;
    int ia[] = { 1, 4, 9, 16 };
    char* s = "hello";
    printf("name c, type char, value %c\n", c);
    printf("name f, type float, value %.3f\n", f);
    printf("name i, type int, value %d\n", i);
    printf("name ia, type int array, value { %d, %d, %d, %d }\n",
```

```
        ia[0], ia[1], ia[2], ia[3]);
printf("name s, type string, value %s\n", s);
return 0;
}
```

5d. Control flow

Write a C function `countOccurrences` that takes two parameters, a string and a character, and returns the number of occurrences of the character in the string. For example, `countOccurrences("hello", 'l')` should return 2.

```
Example solutions
int countOccurrences(char* s, char c) {
    int result=0, i;
    for (i=0; '\0' != s[i]; i++)
        if (s[i] == c)
            result++;
    return result;
}
```

---

<http://www.cs.nyu.edu/courses/summer07/G22.2110-001/hw05-example-solutions.pdf>

Total points: 50.