CSE 374 Final Exam 3/15/17 Sample Solution

Question 1. (8 points) Suppose we have the following two statements in a C program:

```
int *x = malloc(sizeof(int));
int *y = malloc(sizeof(int));
```

For each of the following expressions, write "true" if the expression is *always* true, "false" if the expression is *always* false, or "unknown" if the expression could be either true or false depending on what happens when the program is executed.

__unknown__ `(x < y)`

__unknown__ `(x == y)` **(will be false unless both `malloc`s return `NULL`)**

__unknown__ `(((uintptr_t)x)+sizeof(int))==((uintptr_t)y))`
(Would be false for our memory manger, so we allowed credit for that, but could be true given a different implementation)

__unknown__ `(x == NULL)`

Question 2. (6 points) A little preprocessor magic. What output does this C program produce when it is executed?

```
#include <stdio.h>

#define FUN(x,y,z) x*y+z
#define WORK(a,b) a-b

int main() {
  int a = FUN(2,3,4);
  printf("a = %d\n", a);
  int b = WORK(5, 2+2);
  printf("b = %d\n", b);
  int n = FUN(WORK(5,3), 4, 5);
  printf("n = %d\n", n);
  return 0;
}
```

Output (fill in the numbers):

```
a = 10

b = 5

n = -2
```

Question 3. (8 points) Making things. We have a program that has several header and implementation files, as follows:

```
--------------------
thing.h
--------------------
#ifndef THING_H
#define THING_H
...
#endif
--------------------
stuff.h
--------------------
#ifndef STUFF_H
#define STUFF_H
#include "thing.h"
#include "other.h"
...
#endif
--------------------
other.h
--------------------
#ifndef OTHER_H
#define OTHER_H
...
#endif
```

```
--------------------
one.c
--------------------
#include "stuff.h"
...
--------------------
two.c
--------------------
#include "other.h"
...
--------------------
main.c
--------------------
#include "thing.h"
#include "stuff.h"

int main() { ... }
```

One of last summer's interns tried to put together a Makefile to build a program named `gadget` from these source files, and this is what we've got:

```
one.o: one.c stuff.h thing.h other.h
   gcc -Wall -g -std=c11 -c one.c

two.o: two.c other.h
   gcc -Wall -g -std=c11 -c two.c

main.o: main.c thing.h stuff.h other.h
   gcc -Wall -g -std=c11 -c main.c

gadget: one.o two.o main.o
   gcc -Wall -g -std=c11 -o gadget one.o two.o main.o
```

Fixes in rules are shown above. Also, need to move the last rule (for `gadget`) to the beginning of the Makefile so it becomes the default target.

Unfortunately it doesn't do the job correctly. For starters, when we type `make` with no arguments, it doesn't build the `gadget` executable program for some reason. Also, it doesn't seem to always recompile everything needed after some of the source files have been changed. Show how to fix the problems by writing changes or additions in the `Makefile` code above.

Question 4. (14 points) A trieing question. In hw5 we used a trie to store the words in a dictionary based on the digit sequences that encoded the words. There are several possible ways to represent trie nodes, but for this problem, assume that a node is defined as follows:

```
struct tnode {              // one node in the trie:
  char * word;              //  C-string if this node has a
                            //  word attached, otherwise NULL
  struct tnode *next[10];   //  Subtrees.  next[2] to next[9]
                            //  are subtrees for digits 2-9;
                            //  next[0] is the synonym ('#') link.
};                          //  For 0 <=i<10, next[i]==NULL if
                            //  next[i] is an empty subtree.
```

Complete the following function so that it returns the number of words (strings) stored in the trie with root `r`. In other words, count and return the number of nodes in the trie that have a `word` pointer that is not `NULL`. Hint: recursion is your friend. (Really!)

```
// return number of strings stored in the trie with root r
int nstrings(struct tnode *r) {
  if (r == NULL) {
    return 0;
  }
  int ns = 0;
  if (r->word != NULL) {
    ns++;
  }
  for (int i = 0; i < 10; i++) {
    ns += nstrings(r->next[i]);
  }
  return ns;
}
```

Question 5. (9 points) Testing. This question is about testing the function from the previous question, but you should be able to answer it either before or after writing the code for that question.

Describe three *black-box* tests for the `nstrings` function from the previous question. Remember that a good test specifically describes any setup needed, the test input, if any, and the expected test output. For full credit you must give three distinct tests – i.e., the tests should be different in some significant way, rather than basically testing the same thing, even if the data is slightly different. Try to keep your descriptions brief and to the point.

There are many possibilities. Here are several:

Input: NULL (an empty trie). Verify that `nstrings(NULL)` returns 0.

Input: Create a trie with a single, one-letter word like "a". That trie will have exactly one node in it. Verify that `nstrings` returns 1.

Input: Create a trie with a single word that has multiple characters like "cat". That trie will have multiple nodes, but only one string. Verify that `nstrings` returns 1.

Input: Create a trie with two words that have different digit sequences like "foo" and "bar". Verify that `nstrings` returns 2.

Input: Create a trie with two words that have the same digit sequence like "bat" and "cat". Verify that `nstrings` returns 2. (Note that this is different from the previous test since it will use the synonym link 0 to connect the trie nodes instead of the regular 2-9 digit links.)

Grading note: the description of a good test should be specific enough that someone could figure out how to translate it directly into code. Several answers said things like "create a trie with a bunch of strings, add another, and verify that `nstrings` returns a value that is one larger than before". A good test should describe exactly what strings are in the trie, and what result `nstrings` should return for that trie.

Also, a good test for `nstrings` should be focused on that function. Some of the answers included things like "create a trie with the string 'cat' in it, then try to add 'cat' again; verify that `nstrings` returns the same value." That is primarily a test of `add`, not `nstrings`, since the main role of `nstrings` in that test is to observe if the `add` function worked properly.

Question 6. (12 points) A little debugging. The following C program reads a list of names (lines) from its input file and stores them in a linked list. Once it reaches the end of the file, it frees the linked list and terminates. Unfortunately it contains some sort of memory management errors, as shown in the valgrind output on the next page.

On the listing below, indicate where the errors are located and show how to fix them. You should not change the program logic – it does whatever it does. Just show the changes needed to eliminate the errors reported by valgrind and any other memory management bugs that might be present.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct namelist {
  char* name;
  struct namelist* next;
};
struct namelist* create(char* string) {
  struct namelist* newNode =
           (struct namelist*)malloc(sizeof(struct namelist));
  char* s = (char*)malloc(strlen(string)+1);
  strcpy(s,string);
  newNode->name = s;
  newNode->next = NULL;
  return newNode;
}
int main(int argc, char** argv) {
  char name[100];
  FILE* fp = fopen(argv[1],"r");
  struct namelist* NameList = NULL;
  while(fgets(name,100,fp)) {
    if(NameList == NULL)
      NameList = create(name);
    else {
      struct namelist* front = create(name);
      front->next = NameList;
      NameList = front;
    }
  }
  fclose(fp);
  free(NameList);
  return 0;
}
```

Need to allocate extra byte for '\0'

Close file to free memory allocated by library

Must free all nodes and strings in list, not just first node. Code not required, but this works:

```
struct namelist* p = NameList;
while (p != NULL) {
  struct namelist* next = p->next;
  free(p->name);
  free(p);
  p = next;
}
```

(see next page for valgrind output – do not remove this page from the exam)

Question 6. (cont.) Here is the valgrind output produced when we executed the program on the previous page. Indicate the errors and show the corrections needed to fix them by writing on the program code on the previous page. You may remove this page from the exam while working on the question if you wish.

```
[vasisht@attu2 ~]$ valgrind --leak-check=full ./memleak test_file
==182097== Memcheck, a memory error detector
==182097== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==182097== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==182097== Command: ./memleak test_file
==182097==
==182097== Invalid write of size 1
==182097==    at 0x4C2AAE3: strcpy (vg_replace_strmem.c:506)
==182097==    by 0x4006EA: create (in /homes/iws/vasisht/memleak)
==182097==    by 0x40075B: main (in /homes/iws/vasisht/memleak)
==182097==  Address 0x51f4316 is 0 bytes after a block of size 6 alloc'd
==182097==    at 0x4C27BE3: malloc (vg_replace_malloc.c:299)
==182097==    by 0x4006D3: create (in /homes/iws/vasisht/memleak)
==182097==    by 0x40075B: main (in /homes/iws/vasisht/memleak)
==182097==
==182097== Invalid write of size 1
==182097==    at 0x4C2AAE3: strcpy (vg_replace_strmem.c:506)
==182097==    by 0x4006EA: create (in /homes/iws/vasisht/memleak)
==182097==    by 0x40076D: main (in /homes/iws/vasisht/memleak)
==182097==  Address 0x51f43b6 is 0 bytes after a block of size 6 alloc'd
==182097==    at 0x4C27BE3: malloc (vg_replace_malloc.c:299)
==182097==    by 0x4006D3: create (in /homes/iws/vasisht/memleak)
==182097==    by 0x40076D: main (in /homes/iws/vasisht/memleak)
==182097==
==182097==
==182097== HEAP SUMMARY:
==182097==     in use at exit: 596 bytes in 4 blocks
==182097==   total heap usage: 5 allocs, 1 frees, 612 bytes allocated
==182097==
==182097== 6 bytes in 1 blocks are definitely lost in loss record 2 of 4
==182097==    at 0x4C27BE3: malloc (vg_replace_malloc.c:299)
==182097==    by 0x4006D3: create (in /homes/iws/vasisht/memleak)
==182097==    by 0x40076D: main (in /homes/iws/vasisht/memleak)
==182097==
==182097== 22 (16 direct, 6 indirect) bytes in 1 blocks are definitely lost in
loss record 3 of 4
==182097==    at 0x4C27BE3: malloc (vg_replace_malloc.c:299)
==182097==    by 0x4006BB: create (in /homes/iws/vasisht/memleak)
==182097==    by 0x40075B: main (in /homes/iws/vasisht/memleak)
==182097==
==182097== LEAK SUMMARY:
==182097==    definitely lost: 22 bytes in 2 blocks
==182097==    indirectly lost: 6 bytes in 1 blocks
==182097==      possibly lost: 0 bytes in 0 blocks
==182097==    still reachable: 568 bytes in 1 blocks
==182097==         suppressed: 0 bytes in 0 blocks
==182097== Reachable blocks (those to which a pointer was found) are not shown.
==182097== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==182097==
==182097== For counts of detected and suppressed errors, rerun with: -v
==182097== ERROR SUMMARY: 4 errors from 4 contexts (suppressed: 0 from 0)
```

Question 7. (11 points). Bugs 'R Us. Some of your colleagues are trying to get a program to work, but whenever it runs, it produces a segfault. Since gdb is supposed to be useful for this sort of thing, they tried running the program using gdb, but can't seem to get any useful information from it. Here is what they did get:

```
$ gdb ./bomb
GNU gdb (GDB) Red Hat Enterprise Linux 7.11-67.el7
Reading symbols from ./bomb...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/perkins/cse374/final/bomb
Missing separate debuginfos, use: debuginfo-install glibc-2.17-
157.el7_3.1.x86_64
Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7ab06b6 in __strcpy_sse2_unaligned () from /lib64/libc.so.6
(gdb) bt
#0  0x00007ffff7ab06b6 in __strcpy_sse2_unaligned () from
/lib64/libc.so.6
#1  0x00000000004005ed in process ()
#2  0x0000000000400684 in main ()
(gdb) up
#1  0x00000000004005ed in process ()
(gdb) list
No symbol table is loaded.  Use the "file" command.
(gdb)
```

(a) (3 points) What is the most likely reason that gdb is not able to provide any useful information that would help debug the program?

The `-g` option was probably omitted when the program was compiled and/or linked.

(b) (8 points) Help your colleagues by writing a list of the steps they should do so they can use `gdb` to locate and print the variable or expression that is causing the segfault. If it matters, the source code for the program is contained in a single file `bomb.c`.

- **Recompile and relink the code using the `gcc -g` option.**
- **Start `gdb` as above with `gdb ./bomb` and use `gdb run` to begin execution.**
- **When `gdb` catches the segfault, use `bt` to look at the call stack.**
- **Use `up` to move from any library functions to the immediate calling function in the user's code (`process` in the listing above).**
- **Use `list` or other commands to look at the code and then use `print` or other commands to print variables or expressions to see which one is `NULL` or has an out-of-bounds address that generated the segfault.**

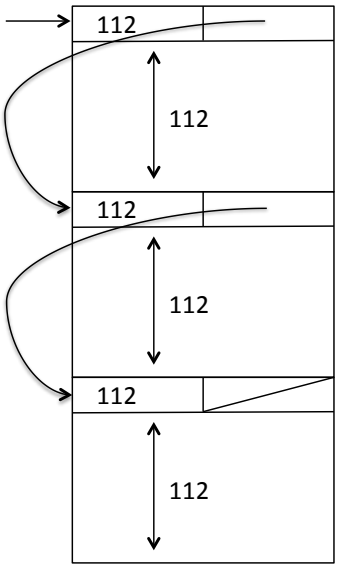
Question 8. (20 points) Memory madness. Sometimes it would be convenient to create a collection of new free blocks all at once in our memory manger package. For this problem, implement a function `make_blocks(nbr, sz)` that returns a pointer to a new set of free blocks that could then be added to the free list. Parameter `nbr` is the number of blocks that we want, and `sz` is the requested number of payload (free) bytes in each block, not including the block headers. The function should round `sz` up to the next multiple of 16, then call `malloc` to obtain enough new storage for the requested blocks and their headers, then "split" this new block into `sz` smaller blocks by inserting appropriate block header nodes in the new space and linking them together. Finally the function should return a pointer to this new list of blocks.

We assume that each free block has the following header:

```
struct free_node {
  uintptr_t size;           // # of payload bytes
  struct free_node * next;  // next node on free list or NULL
};
```

For this problem, we will assume that the `size` field in the header contains only the size of the payload, not including the size of the header itself.

So, for example, if we call `make_blocks(3, 100)`, the function should round the block size up to 112 (= 7 * 16), then allocate 3*(16+112) bytes to include space for the three blocks plus their headers, create `free_node` structs for each new free block inside the newly allocated space, link those header structs together, and finally return a pointer to the first of these block headers. The result should look like this:



Complete the function on the next page so it works as specified above. You may remove this page for reference.

Question 8. (cont.) Provide an implementation of `make_blocks` below. An additional blank page is provided after this one if you need more space. You do not need to provide extensive comments, but a few comments that indicate what is happening in each major section of the code would be helpful for reader and grader sanity.

```
// Allocate space for nbr blocks each with room for a sz byte
// payload, rounded up to a multiple of 16; then turn the space
// into a linked list of nbr blocks, and return a pointer to the
// header of the first block. Assume that nbr, sz > 0, and the
// function does not need to check these parameter values.
// Return NULL if allocation is unsuccessful.
struct free_node * make_blocks(uintptr_t nbr, uintptr_t sz) {
  // Round size up to a multiple of 16
  sz = ((sz+15)/16)*16;

  // Allocate nbr blocks with room for headers and sz payload.
  // Return NULL if not successful.
  // (assumes sizeof(struct free_node) is a multiple of 16)
  struct free_node * block = (struct free_node *)
                malloc(nbr * (sz + sizeof(struct free_node)));
  if (block == NULL)
    return NULL;

  // Initialize free_node headers for all but last block
  struct free_node *current = block;   // current header location
  for (int i = 0; i < nbr-1; i++) {
    current->size = sz;
    current->next = (struct free_node *)
         (((uintptr_t)current) + sz + sizeof(struct free_node));
    current = current->next;
  }

  // fix up header in last block
  current->size = sz;
  current->next = NULL;

  return block;
}
```

Other correct answers also received full credit, of course.

Question 9. (14 points) The traditional, somewhat annoying, C++ virtual function question. Consider the following program, which compiles and executes without error.

```
class Base {
public:
  virtual void f() { g(); cout << "tweet" << endl; }
          void g() { h(); cout << "chirp" << endl; }
  virtual void h() {      cout << "warble" << endl; }
};

class Thing: public Base {
public:
  void g() { h(); cout << "grunt" << endl; }
  void h() {      cout << "glorp" << endl; }
};

int main() {
  Base *b = new Base();
  b->f();
  cout << "---" << endl;
  Thing *t = new Thing();
  b = t;
  b->f();
  cout << "---" << endl;
  t->f();
  cout << "---" << endl;
  b->g();
  cout << "---" << endl;
  t->g();

  return 0;
}
```

In the box to the right, write the output that is produced when this program is executed.

Output:

```
warble
chirp
tweet
---
glorp
chirp
tweet
---
glorp
chirp
tweet
---
glorp
chirp
---
glorp
grunt
```

Question 10. (8 points) Concurrency. Suppose we have the following global variable and function in a C program:

```
int num = 0;  // num == 0 when program starts

void bump() {
  int temp = num;
  temp = temp + 1;
  num = temp;
}
```

Now, suppose we have two threads executing concurrently. The threads use the same memory that contains variable `num` and both can call function `bump`. Both threads call function `bump` twice, and no other code calls `bump` or accesses the variable `num`:

Thread 1	Thread 2
`bump(); bump();`	`bump(); bump();`

(a) (4 points) Circle all of the possible values that variable `num` could have after both threads finish. You should assume that each thread executes its statements in order, and that memory (variable) reads and writes work as written (i.e., no funny out-of-order reads and writes due to caches or physical memory implementation). However you should not make any assumptions about which thread runs first or whether the threads execute in some interleaved order or even simultaneously on different processors.

Possible final values for `num` (circle all possible): 0 1 (2 3 4) 5 6 or more

(b) (4 points) We think the programmer intended for the final value of `num` to contain the total number of times that `bump` is called. What could we do to guarantee that this is true even if `bump` is called by code in several different threads? Give a brief but technical description. If you wish, you can show how to modify the above code to guarantee that it will work as intended when executed concurrently. However, if, in fact, the code is already thread-safe (i.e., always produces the same result), give a short technical explanation of why that is the case.

`bump` needs to be fixed so the three assignment statements in it are executed as an single atomic unit (critical section). If our language provided something like the hypothetical `atomic{...}` construct from lecture, we could use that to surround the statements in `bump`. However, in most languages we would need to create a lock variable to control access to `num`, then acquire the lock at the very beginning of `bump` and release it just before returning to the caller.

Grading note: Several answers said things like "use a lock" or "use `atomic`". Those answers received partial credit, but for full credit, an answer needed to describe what to lock or surround with `atomic` to avoid the race condition.

Have a great spring break! The CSE 374 Staff