CSE 333 Midterm Exam Sample Solution 7/28/14

Question 1. (20 points) C programming. For this question implement a C function `contains` that returns 1 (true) if a given C string appears as a substring of another C string starting at a given position.

Examples: The following should return 1 (true): `contains("abc","abc",0); contains ("bc","abc",1); contains("xyz","xyzzy",0).` The following calls return 0 (false): `contains("abc","xyz",1); contains("abc","abcde",1); contains("def","abcde",3); contains("de","abcde",1).`

Restriction: the only function you may use from the `<string.h>` library is `strlen`.

Simplifying assumptions: Assume that pointer arguments are non-null and point to properly `'\0'`-terminated C strings that have at least one character in them besides the terminating `'\0'`. You may also assume that `pos` is non-negative (i.e, `pos>=0`).

```
// return 1 if string s appears in target starting at pos

int contains (char *s, char *target, int pos) {
  // return false immediately if s runs past end of
  // target starting at loc
  if (pos + strlen(s) > strlen(target))
    return 0;

  // Compare characters of s to target+pos.  If they differ
  // then the strings don't match.  If we get to the \0
  // byte at the end of s, then everything matched
  char *sptr = s;
  char *tptr = target+pos;
  while (*sptr != '\0') {
    if (*sptr != *tptr)
      return 0;
    sptr++;
    tptr++;
  }

  // all of s examined and no mismatch
  return 1;
}
```

There are certainly many other solutions, including ones that look for a terminating ‘\0’ in both strings simultaneously, but there still needs to be some sort of check to be sure that `pos` is not so large that `target+pos` references beyond the ‘\0’ at the end of `target`.

Question 2. (12 points) Here is a small header file and a main program that includes it.

File `midterm.h`

```
#ifndef _MIDTERM_H_
#define _MIDTERM_H_

#define NEW_PI PI+1
#define GOLDEN 1.618
#define NEWER_PI(x) (x*PI)

typedef int number;

#endif  // _MIDTERM_H_
```

File `midterm.c`

```
#include "midterm.h"

#define PI 3.1415
#define GOLDEN 1.618+1

int main(int argc, char **argv) {
  number x = PI;
  int xx = NEW_PI;
  int y = GOLDEN;
  int z = NEWER_PI(y+1);
  return 0;
}
```

Below, write the output produced by the C preprocessor (`cpp` or `gcc -E`) showing exactly how file `midterm.c` is rewritten and sent to the compiler phase of `gcc`. (The preprocessor can correctly process this file without any blocking errors.)

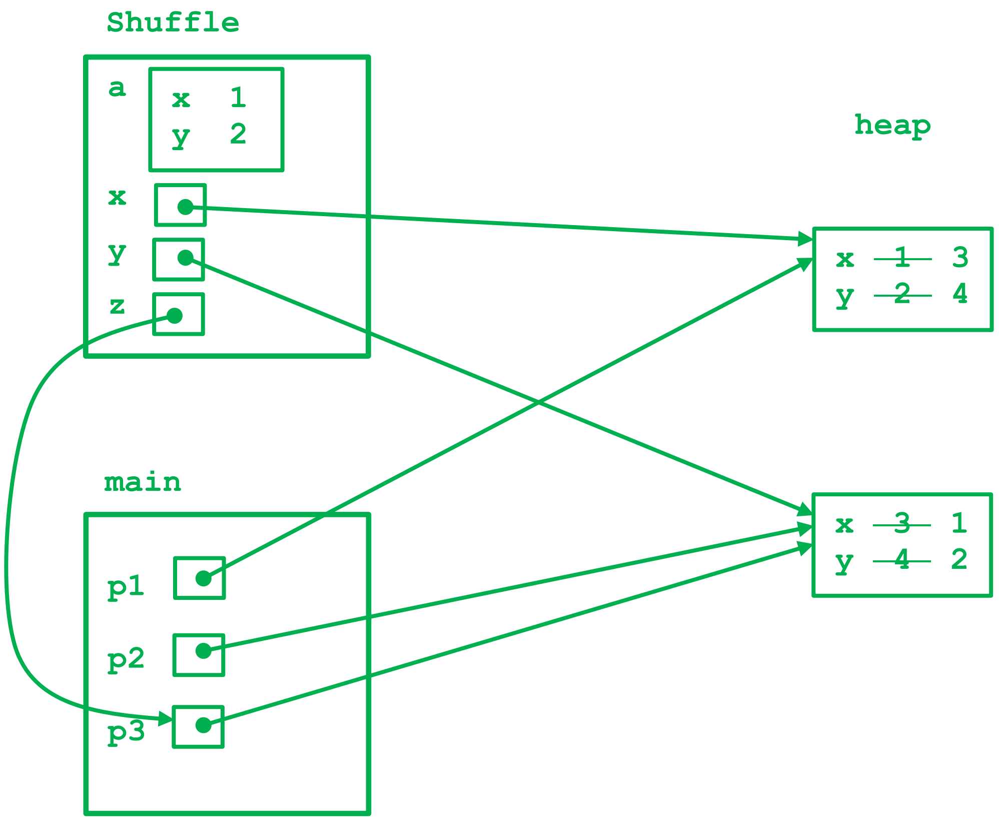
```
typedef int number;

int main(int argc, char **argv) {

  number x = 3.1415;

  int xx = 3.1415 +1;

  int y = 1.618+1;

  int z = (y+1*3.1415);

  return 0;

}
```

Question 3. (24 points) The somewhat traditional, always annoying C program. Consider the following program, which does compile and execute without any errors. (We assume for this question that `malloc` always succeeds and returns a non-null pointer.)

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

typedef struct point2d {  // 2-d point
  int x, y;
} Point2d, *Point2dPtr;

void PrPoint(char *label, Point2dPtr p) {
  printf("%s %d %d\n", label, p->x, p->y);
}

Point2dPtr NewPt(int a, int b) {
  Point2dPtr p = (Point2dPtr)malloc(sizeof(Point2d));
  p->x = a;
  p->y = b;
  return p;
}

void Shuffle(Point2d a,Point2dPtr x,Point2dPtr y,Point2dPtr *z) {
  *z = y;
  *x = *y;
  **z = a;
  //// HERE ////
}

int main() {
  Point2dPtr p1 = NewPt(1, 2);
  Point2dPtr p2 = NewPt(3, 4);
  Point2dPtr p3;
  Shuffle(*p1, p1, p2, &p3);
  PrPoint("p1", p1);
  PrPoint("p2", p2);
  PrPoint("p3", p3);
  return 0;
}
```

Answer questions about this program on the next page. You may remove this page for reference if you wish.

Question 3. (cont.) (a) (15 points) Draw a boxes 'n arrows diagram showing state of memory when control reaches the comment containing `//// HERE ////` in the middle of function `Shuffle`. Your diagram should have boxes showing the stack frames for all active functions. The stack frames should show values of all local variables. Draw each `point2d struct` as a box with two labeled fields `x` and `y`. Draw an arrow from each pointer to the location that it references. Data that is allocated on the heap should be drawn in a separate area, since it is not part of any function stack frame After drawing your diagram, be sure to answer part (b) at the bottom of the page.



(b) (9 points) What output does this program produce when it is executed?

```
p1 3 4
p2 1 2
p3 1 2
```

Question 4. (24 points) Here is an attempt to fix the buggy vector program. Be sure to read the rest of the question before tearing out any pages or doing any work. ...

```c
#include <stdbool.h>      // for bool
#include <stdlib.h>       // for malloc, free

#define INIT_VEC_SIZE 8
#define FINAL_VEC_SIZE 10

typedef void* element_t;

typedef struct vector_t {
  size_t length;    //  length of arry
  element_t *arry;  //  data items - unused entries are NULL
} *vector_t;

// Fill the array in v with the elements 0 to n-1
// Return true if it succeeds
bool FillElements(size_t n, vector_t v);

// Create a vector and then add some values to the vector array
int main(int argc, char *argv[]) {

  // allocate initial vector and exit if failure
  vector_t v = (vector_t)malloc(sizeof(struct vector_t));
  if (v == NULL)
    return EXIT_FAILURE;
  v->arry = (element_t*)malloc(INIT_VEC_SIZE *sizeof(element_t));
  if (v->arry == NULL) {
    free(v);
    return EXIT_FAILURE;
  }

  // save length in vector struct and initialize array elements
  v->length = INIT_VEC_SIZE;
  bool success = FillElements(FINAL_VEC_SIZE, v);
  if(!success)
    return EXIT_FAILURE;
}

// fill the array in v with the elements 0 to n-1
// return true if successful
bool FillElements(size_t n, vector_t v){
  size_t i;

  // store new int objects in so vector[i] holds int i.
  // exit if any step fails.
  for (i = 0; i < FINAL_VEC_SIZE; ++i) {
    int *x = (int*)malloc(sizeof(int));
    if (x == NULL)
      return false;
    *x = i;
    v->arry[i] = x;
  }
  return true;
}
```

Question 4. (cont.) There seem to be some memory problems, so we ran this program using valgrind. Here's the output we got:

```
==9424== Memcheck, a memory error detector
==9424== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==9424== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==9424== Command: ./buggy
==9424==
==9424== Invalid write of size 8
==9424==    at 0x400671: FillElements (buggy.c:50)
==9424==    by 0x4005FB: main (buggy.c:33)
==9424==  Address 0x4c2a0d0 is 0 bytes after a block of size 64 alloc'd
==9424==    at 0x4A06409: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==9424==    by 0x4005B4: main (buggy.c:25)
==9424==
==9424==
==9424== HEAP SUMMARY:
==9424==     in use at exit: 120 bytes in 12 blocks
==9424==   total heap usage: 12 allocs, 0 frees, 120 bytes allocated
==9424==
==9424== 8 bytes in 2 blocks are definitely lost in loss record 1 of 4
==9424==    at 0x4A06409: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==9424==    by 0x40063B: FillElements (buggy.c:46)
==9424==    by 0x4005FB: main (buggy.c:33)
==9424==
==9424== 112 (16 direct, 96 indirect) bytes in 1 blocks are definitely lost in loss record 4 of 4
==9424==    at 0x4A06409: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==9424==    by 0x400598: main (buggy.c:22)
==9424==
==9424== LEAK SUMMARY:
==9424==    definitely lost: 24 bytes in 3 blocks
==9424==    indirectly lost: 96 bytes in 9 blocks
==9424==      possibly lost: 0 bytes in 0 blocks
==9424==    still reachable: 0 bytes in 0 blocks
==9424==         suppressed: 0 bytes in 0 blocks
==9424==
==9424== For counts of detected and suppressed errors, rerun with: -v
==9424== ERROR SUMMARY: 4 errors from 3 contexts (suppressed: 2 from 2)
```

Your job is to find all of the bugs in the program and indicate how to fix them. Use the information in the program itself as well as the valgrind output to locate the bugs (and it may be that there are some bugs that don't result in any error report from valgrind, so be sure to check the source code carefully).

You can either mark the original program to show what's wrong and how to fix it, or you can write an explanation on the following page. Feel free to detach this page (with the valgrind output) while you work, but be sure to turn in the page with the program code if you mark your answers there.

Question 4. (cont.) You can use the space on this page for your answer if you wish, or you can mark your answers on the program code. Just be sure the grader can follow what you've done to give you as much credit as possible.

Here are the bugs that were deliberately introduced into the code. We tried to allow generous partial credit, particularly if you missed subtle points.

- **If the `malloc` in `FillElements` fails, any elements of the array that were previously allocated need to be freed before returning `false`**
- **The elements of the array, the array, and the vector need to be freed at the end of `main`**
- **If `FillElements` returns `false`, the array and vector need to be freed in `main`**
- **In order to fill `n` elements, the array needs to be resized if it is not large enough. After freeing the old array and allocating space for the new array (`malloc` or `realloc`), there should be a check to see if the allocation succeeds and return `false` if it doesn't. If allocation succeeds, `v->length` needs to be updated.**
- **in `FillElements`, unused elements of the array need to be initialized to `NULL` if `n` is less than the size of the array.**
- **In `FillElements`, the `for` loop initializes `FINAL_VEC_SIZE` elements. It should be `n`.**
- **The end of `main` is missing `return EXIT_SUCCESS`.**

An additional, subtle point: Many people felt there was an error in this line of code:

```
  vector_t v = (vector_t)malloc(sizeof(struct vector_t));
```

The subtlety is that the name `vector_t` has two distinct meanings in the code. One is as a `struct` tag name, while the other is a `typedef` name that is a synonym for `struct vector_t *` (i.e., pointer to `struct vector_t`). These two definitions are distinct and it depends on context to decide which one applies.

So in the above code, the first two uses of `vector_t` are the pointer type, while the use of `vector_t` in the `sizeof` expression is the `struct` tag, and it is correct. Changing the code so it reads `malloc(sizeof(vector_t))` would allocate space sufficient to store a pointer, but not enough for the `struct`.

Question 5. (20 points) A little C++ programming. We've been having so much fun with linked lists, it seemed a shame not to do a bit more in C++. Here is the header for a class that implements a list of C++ `string`s using a single-linked list made up from `StringNode` objects.

```
#include <string>
using namespace std;

class StringNode {     // single node in a linked list of strings
 public:
  string value_;       // value in this node
  StringNode * next_; // next node in the list or nullptr if none

  // convenience constructor - initialize new node with string s
  // and next pointer p
  StringNode(string s, StringNode *p): value_(s), next_(p) { }
};


class StringLinkedList {
 public:

  // construct empty StringLinkedList
  StringLinkedList() : first_(nullptr), last_(nullptr) { }


  // additional operations omitted...


 private:
  StringNode * first_;   // first node in list
  StringNode * last_;    // last node in list
                         // first_ = last_ = nullptr if the
                         // list is empty.
};
```

You should assume that all dynamic memory allocation and deletion functions will succeed without errors – you do not need to check for failures or handle exceptions.

You can remove this page for reference while working on the question if you wish.

Question 5. (cont.) We would like to add a += assignment operator to this class to append new strings to a list. If `lst` is an object of type `StringLinkedList`, then the operation `lst+=s` should create a new node holding the C++ `string` `s` and add it (append it) as the last node of the list `lst`.

(a) (5 points) Give the declaration for this new += operation to be added to the declaration of class `StringLinkedList` (above).

```
     StringLinkedList &operator+=(const string &s);
```

(b) (15 points) Give the full implementation of this added += operation as it would appear in a separate source file `StringLinkedList.cc`. Your answer should contain any necessary `#include` or other compiler and preprocessor directives needed.

```
#include "StringLinkedList.h"
#include <string>
using namespace std;

StringLinkedList &StringLinkedList::operator+=(const string &s) {
  // allocate new node
  StringNode * p = new StringNode(s,nullptr);

  // if list is empty this is now the first node
  if (first_ == nullptr) {
    first_ = last_ = p;
  } else {
    // add to end of list and link to former last node
    last_->next_ = p;
    last_ = p;
  }

  // return reference to this StringLinkedList
  return *this;
}
```

Some solutions omitted `#include <string>` or `using namespace std;`, probably assuming that since these were already found in the `StringLinkedList` header they would be indirectly included anyway. But every file should normally `#include` all header files that it references, both for documentation and to avoid problems if some of the headers it uses are later changed and no longer `#include` the same files or have the same `using` directives that they did previously.