CSE 333 Midterm Exam 5/6/16 Sample Solution

Question 1. (16 points) Making things. Suppose we have the following C header and implementation files, showing the various `#include` and `#if…` directives in each one:

```
======
blob.h
======
#ifndef _BLOB_H_
#define _BLOB_H_
...
#endif

======
blob.c
======
#include "blob.h"
#include "thing.h"
...

======
thing.h
======
#ifndef _THING_H_
#define _THING_H_
#include "blob.h"
...
#endif
```

```
======
thing.c
======
#include "thing.h"
...

======
gadget.c
======
#include "blob.h"
...

======
widget.c
======
#include "blob.h"
...
```

On the next page, write an appropriate `Makefile` that will compile all of this code and link it together to produce an executable program named `widget`. To compile and link the code, use `gcc` with the `-Wall`, `-g` and `-std=c11` options. Your `Makefile` should only recompile or relink files if needed to bring targets up to date. The default target in the `Makefile` should be the executable program `widget`, i.e., if `make` is run with no arguments, the `widget` program (target) should be built. In case it matters, the `main` function is in file `widget.c`.

Hint: the gcc option `-c` will compile a source file and stop after producing the corresponding `.o` file.

Use the space on the next page to write your answer. You may remove this page from the exam if you wish.

Question 1. (cont.) Write your `Makefile` in the space below.

```
widget: blob.o thing.o gadget.o widget.o
      gcc -Wall -g -std=c11 -o widget \
                              blob.o thing.o gadget.o widget.o

blob.o: blob.c blob.h thing.h
      gcc -Wall -g -std=c11 -c blob.c

thing.o: thing.c blob.h thing.h
      gcc -Wall -g -std=c11 -c thing.c

gadget.o: gadget.c blob.h
      gcc -Wall -g -std=c11 -c gadget.c

widget.o: widget.c blob.h
      gcc -Wall -g -std=c11 -c widget.c
```

Note: The '\' used to continue the first line actually does work in a Makefile, but we did not expect to see that in any of the answers. Most people either wrote everything on one line or it was clear what was intended if it things didn't quite fit.

Question 2. (20 points) C programming! This question involves the data structures from the HW1 and HW2 projects. Copies of the `LinkedList.h`, `HashTable.h`, and `HashTable_priv.h` header files have been provided on separate pages.

For this problem give the implementation of a new function `HTFindValue` to add to our `HashTable` implementation that will search a hash table and report whether a particular value is found in some key-value pair stored in the `HashTable` and, if the value is found, return the corresponding key. Because users can store arbitrary information in a `HashTable`, the `HTFindValue` function has a parameter that is itself a function that compares two `HTValue_t` values and reports whether they are equal or not. Here are the specifications for the comparison function and of `HTFindValue`:

```
// Compare two HTValue_ts and report whether they are equal
// Arguments:
// - val1: the first value to be checked for equality
// - val2: the second value to be checked for equality
//
// Returns:
// - +1 (true) if val1 is equal to val2; 0 (false) otherwise
//
typedef bool(*ValueEqualsFnPtr)(HTValue_t val1,
                                HTValue_t val2);

// Search a HashTable for a specified value and return the
// associated key if found.
//
// Arguments:
// - ht: the HashTable to search.
// - value: the value to search for.
// - value_equals_function: pointer to a function that compares
//   two HTValue_t items and returns 1 if equal and 0 if not.
// - key: if the value is found, an associated key is returned
//   to the caller via this return parameter.  If the value is
//   found in more than one key-value pair, one of those keys
//   is returned, but which one is not specified.  If value is
//   not found, the value of this parameter is not specified.
//
// Returns:
//  -1 if there was any error (e.g., out of memory)
//   0 if the value wasn't found in the HashTable
//  +1 if the value was found, and therefore the associated
//     key was returned to the caller via the return parameter.
//
int HTFindValue(HashTable ht,
                HTValue_t value,
                ValueEqualsFnPtr value_equals_function,
                HTKey_t* key);
```

(continued on next page – you may remove this page if you wish)

Question 2. (cont.) Here is some sample code that shows how this function could be used to search for a value in a `HashTable` and recover the associated key if present:

```
typedef struct {
    int num;
} ExampleValue, *ExampleValuePtr;

bool ExampleValueEqualsFn(ExampleValuePtr val1,
                          ExampleValuePtr val2) {
    return val1->num == val2->num;
}

// search for value 333 in table ht and print results
void contains333(HashTable ht) {
    ExampleValue ev;
    ev.num = 333;

    HTKey_t key;
    int res = HTFindValue(ht, (HTValue_t) &ev,
                (ValueEqualsFnPtr) ExampleValueEqualsFn, &key);

    if (res == 1)
        printf("found value 333 with key %" PRIx64 "\n", key);
    else if (res == 0)
        printf("does not contain value 333\n");
    else
        printf("error!\n");
    return;
  }
```

Your answer may use any of the functions or data declared in the `LinkedList.h`, `HashTable.h`, and `HashTable_priv.h` headers. Don't be alarmed if the solution turns out to be fairly short.

Write your answer on the next page. You may remove this page and the previous one from the exam if you wish.

Question 2. (cont.) Write your implementation of function `HTFindValue` below (see specification on previous pages).

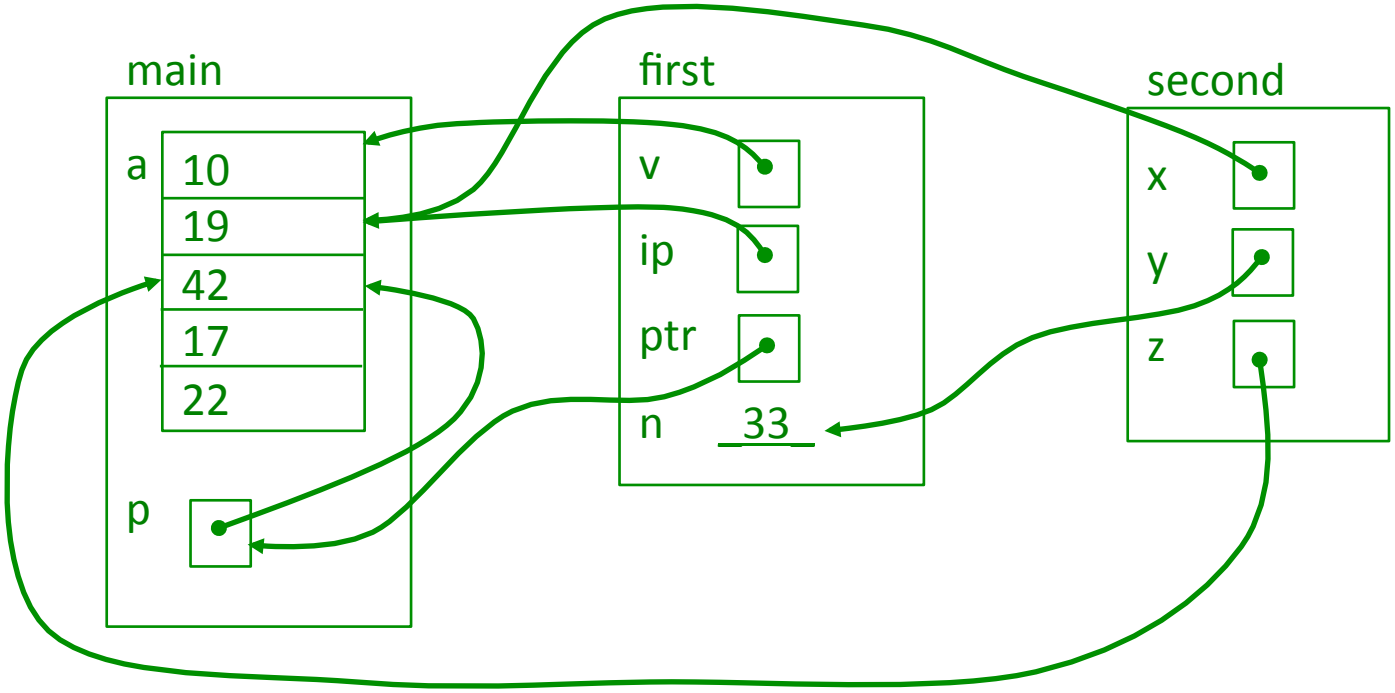
```
int HTFindValue(HashTable ht,
                HTValue_t value,
                ValueEqualsFnPtr value_equals_function,
                HTKey_t* key) {

  HTIter it = HashTableMakeIterator(ht);
  if (it == NULL)
    return -1;

  HTKeyValue item;
  while (!HTIteratorPastEnd(it)) {
    if (HTIteratorGet(it, &item) != 1) {
      HTIteratorFree(it);
      return -1;
    }
    if (value_equals_function(item.value, value)) {
      *key = item.key;
      HTIteratorFree(it);
      return 1;
    }
    HTIteratorNext(it);
  }
  HTIteratorFree(it);
  return 0;

}
```

Question 3. (22 points) Pointers and arrays. Consider the following program, which, as is traditional, compiles and executes with no warnings or errors:

```
#include <stdio.h>

// print title then n integers starting at *a, followed by \n
void pr(char* title, int *a, int n) {
  printf("%s:", title);
  for (int i=0; i<n; i++) {
    printf(" %d", a[i]);
  }
  printf("\n");
  return;
}

void second(int *x, int *y, int *z) {
  z[2] = 22;
  *y = 33;
  printf("*x = %d, *y = %d, *z = %d\n", *x, *y, *z);
  //// HERE ////
  return;
}

void first(int *v, int *ip, int **ptr) {
  int n = 0;
  ip[2] = 17;
  v[1] = 19;
  (*ptr)++;
  **ptr=42;
  second(ip, &n, *ptr);
  printf("n = %d\n", n);
}

int main() {
  int a[5] = { 10, 11, 12, 13, 14 };
  int *p = &a[1];

  pr("start", a, 5);
  first(a, p, &p);
  pr("finish", a, 5);
  printf("*p = %d\n", *p);
  return 0;
}
```

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

Question 3. (cont.) (a) (14 points) Draw a boxes 'n arrows diagram showing the memory layout and contents at the point just after the `printf` function call in `second` is executed (marked with `//// HERE ////` in the code). Be sure your diagram clearly shows the values of all variables in all active functions and has a separate box (i.e., stack frame) for each active function. For each pointer, draw an arrow from the pointer to the variable that it references. After finishing your diagram be sure to answer part (b) of this question at the bottom of the page.



(b) (8 points) What does this program print when it is executed?

```
start: 10 11 12 13 14
*x = 19, *y = 33, *z = 42
n = 33
finish: 10 19 42 17 22
*p = 42
```

Question 4. (20 points) Copying things. The function on the next page copies bytes from one stream (file) to another using the POSIX I/O functions. But it's only partially implemented. The part of the function that reads the data is done but the part that writes the data is missing. Your job is to complete the function so it works as specified using only POSIX I/O functions (*not* the C standard I/O library functions like `printf` and `fwrite`).

The specification of the function is as follows:

```
// Copy data from a source stream to a destination stream.
// The client is responsible for providing valid, open file
// descriptors that can be successfully read and written.
// This function does not close the streams before it returns.
// It is up to the client to do whatever it wants with the
// streams after the function returns.
//
// Arguments:
// - src_fd: source file descriptor.  Data is read from here.
// - dst_fd: destination file descriptor.  Data is written here.
//
// Returns 0 if the copy succeeds or -1 if there is an error.
//
int CopyData(int src_fd, int dst_fd);
```

For reference, here is a summary of some key POSIX I/O functions.

`int open(const char *`*name*`, int `*mode*`);`
  mode is one of `O_RDONLY`, `O_WRONLY`, `O_RDWR`
`int creat(const char *`*name*`, int `*mode*`);`
  create a new file
`int close(int `*fd*`);`
`ssize_t read(int `*fd*`, void *`*buffer*`, size_t `*count*`);`
  returns # bytes read or 0 (eof) or -1 (error)
`ssize_t write(int `*fd*`, void *`*buffer*`, size_t `*count*`);`
  returns # bytes written or -1 (error)

You can remove this page from the exam if you wish.

Question 4. (cont.) Complete the implementation of function `CopyData` below.

```
#define SIZE ..  // SIZE is some suitable positive integer

int CopyData(int src_fd, int dst_fd) {
  char buf[SIZE];
  ssize_t rlen, wlen, total;

  do {
    rlen = read(src_fd, buf, SIZE);
    if (rlen == -1 && errno == EINTR) {
      continue;
    } else if (rlen == -1) {
      return -1;
    }

    // add code below to write the data that was just read
    // from src_fd to the dst_fd output stream

    total = 0;
    while (total < rlen) {
      wlen = write(dst_fd, buf + total, rlen - total);
      if (wlen == -1 && errno == EINTR) {
        continue;
      } else if (wlen == -1) {
        return -1;
      }
      total += wlen;
    }

  } while (rlen != 0);
  return 0;
}
```

Question 5. (22 points) Lists in C++. In our continued quest to fix and debug things, we've found a C++ class that implements lists of integers using a single-linked-list data structure. Here's the header file `IntList.h`:

```
#ifndef _INTLIST_H_
#define _INTLIST_H_

#include <iostream>
using namespace std;

// Integer list class implemented with a linked list
class IntList {
 public:
  // construct new empty IntList
  IntList() : first_(nullptr), last_(nullptr) { }

  // IntList destructor
  ~IntList();

  // copy constructor and assignment not supported
  IntList(const IntList&) = delete;
  IntList& operator=(const IntList&) = delete;

  // append integer n to end of this IntList
  void push_back(int n);

  // print this list on stream out as "[v1, ..., vn]"
  friend std::ostream &operator<<(std::ostream &out,
                                  const IntList &lst);

 private:
  // IntList node data structure and instance variables
  struct Node {    // one node in the list:
    int val;       //   node value (payload)
    Node * next;   //   next node in list or nullptr if none
  };

  Node *first_; // first node in list or nullptr if list is empty
  Node *last_;  // last node in list or nullptr if list is empty
};

#endif // _INTLIST_H_
```

(continued on next page – remove this one if you want)

Question 5. (cont.) Here are the parts of the implementation we've been able to find in file `IntList.cc`:

```
#include "IntList.h"
#include <iostream>
using namespace std;

// append integer n to end of this IntList
void IntList::push_back(int n) {
  // need to get around to implementing this some day
}

// destructor
IntList::~IntList() {
  first_ = nullptr;
  last_  = nullptr;
}

// stream output for IntList
// format is [v1, ..., vn]
std::ostream &operator<<(std::ostream &out, const IntList &lst) {
  IntList::Node * p = lst.first_;
  out << "[";
  while (p != nullptr) {
    if (p != lst.first_)
      out << ", ";
    out << p->val;
    p = p->next;
  }
  out << "]";
  return out;
}
```

All of the code on this page and the previous one does compile successfully without any error or warning messages.

Answer questions about this class on the next page. You can remove this page from the test if you want.

Question 5. (cont.) (a) (10 points) Give an implementation of function `push_back`, which adds a link with the new integer as the last node in the linked list.

```
// append integer n to end of this IntList
void IntList::push_back(int n) {

  Node * p = new Node();
  p->val = n;
  p->next = nullptr;
  if (first_ == nullptr) {
    // add first node to previously empty list
    first_ = last_ = p;
  } else {
    // append new node to end of list
    last_->next = p;
    last_ = p;
  }

}
```

(continued on next page)

Question 5. (cont.) (b) (6 points) After implementing the `push_back` function, we've discovered that this `IntList` class has some sort of memory management problems. Sometimes programs using this class seem to run fine, other times they can show memory leaks and/or other problems. Exactly what is(are) the bug(s)? (Give a specific, technical explanation.)

The destructor doesn't work properly. As written, it sets `first_` and `last_` to `nullptr` without deleting any of the nodes in the list, so all of the memory occupied by the nodes is leaked.

(c) (6 points) Show how to fix the bug(s) identified in part (b). If code needs to be deleted, added, or replaced, write down the corrected C++ code below and indicate where it fits in the class. Your answer should be C++ code and specific details about what to change; not just a general verbal description. Use only ordinary pointers in any new or modified code, do *not* use C++smart pointers like `unique_ptr`.

Replace the body of the destructor with code that deletes all of the nodes in the list:

```
IntList::~IntList() {
  // delete list nodes
  Node *p = first_;
  while (p != nullptr) {
    Node *nxt = p->next;
    delete p;
    p = nxt;
  }
}
```