CSE 333 Midterm Exam Nov. 3, 2017 **Sample Solution**

Question 1. (8 points) Making things. Consider this (buggy) `Makefile` (the options `-Wall`, `-g`, and `-std=c11` were omitted to save space and do not affect the answers):

```
all: prog
foo.c: foo.c foo.h bar.h
     gcc -c -o foo.o foo.c
bar.c: bar.c foo.h bar.h
     gcc -c -o bar.o bar.c
prog: foo.o bar.o
     gcc -o prog foo.o bar.o
clean:
     rm prog *.o
```

For the following questions, be clear about which files already exist or not, precisely why the particular situation occurs, etc., and assume that `make` does *not* have any special knowledge about compiling C beyond the rules given in this `Makefile`. There are multiple correct answers to some of the questions; you only need to give a single correct one. Be brief and to the point.

(a) (3 points) Describe a situation where this `Makefile` would lead `make` not to recompile or relink something that needs recompiling or relinking.

If `foo.o` and `bar.o` exist but `prog` does not, and one of the C source files has changed, the `Makefile` will relink `prog` from the old `.o` files without recompiling ones that need updating.

(b) (3 points) Describe a situation where this `Makefile` would lead `make` to report that it does not know about some target.

If `foo.o` or `bar.o` do not exist, running `make` will need those files to be made, but there are no targets to do it.

(c) (2 points) What is wrong with this `Makefile`? How should it be fixed?

The targets `foo.c` and `bar.c` should be `foo.o` and `bar.o`.

Question 2. (14 points) We have a file `ppro.c` containing the following C code:

```
#include <stdio.h>
#define INCR 2
#define PRINT(x) printf("%d\n", x )
#define COMPUTE(x)   (x * x + INCR)
int compute(int x ) { return ( x * x + INCR ); }
int main() {
  int a = 2;
  int b = 3;
  PRINT( compute(a) );
  PRINT( COMPUTE(a) );
  PRINT( compute(a+b) );
  PRINT( COMPUTE(a+b) );
  return 0;
}
```

(a) (10 points) Show the result produced by the C preprocessor when it processes file `ppro.c` (i.e., if we were compiling this file, what output would the preprocessor send to the C compiler that actually translates the program to machine code?). You should ignore the `#include <stdio.h>` directive since that includes library declarations that we do not have access to. Write the rest of the preprocessor output below.

```
int compute(int x ) { return ( x * x + 2 ); }
int main() {
  int a = 2;
  int b = 3;
  printf("%d\n", compute(a) );
  printf("%d\n", (a * a + 2) );
  printf("%d\n", compute(a+b) );
  printf("%d\n", (a+b * a+b + 2) );
  return 0;
}
```

(b) (4 points) What output does this program print when it is executed? (It does compile and execute without errors.)

```
6
6
27
13
```

Question 3. (16 points) A simple C programming warmup. Give an implementation of the following function, which has an array as a parameter and allocates and returns through an output parameter a new array of the same length. In the new array, element *k* should be the sum of elements 0 through *k* of the input array. Example: if the original array contains { 1, 3, 4, 5 }, the new output array should contain { 1, 4, 8, 13 }.

```
#include <stdlib.h>

// Allocate and return a new array where the kth
// element of the new array is the sum of elements
// 0 through k of the original array
// Arguments:
// - arr: the original array
// - arrlen: number of elements in arr
// - outputarr: return parameter - the address of
//   the newly allocated array is stored here.
// Returns true (1) on success, otherwise false
int RunningSum(int *arr, size_t arraylen, int **outputarr) {

  // attempt to allocate the new array
  int *newarr = (int *)malloc(arraylen*sizeof(int));
  if (newarr == NULL)
    return 0;

  // store running product in new array
  newarr[0] = arr[0];
  for (int i = 1; i < arraylen; i++) {
    newarr[i] = newarr[i-1] + arr[i];
  }

  // return new array and indicate success
  *outputarr = newarr;
  return 1;
}
```

During the exam we announced that for this problem it would be fine to assume `arraylen>0`. In production software we would want to either check for that with an `assert` (after modifying the specification to require it), or fix the code so it did not reference any elements of the array if `arraylen` is 0.

Question 4. (20 points) Making hash out of pointers. This question concerns data structures from the HW1 project. Copies of the header files for `LinkedLists` and `HashTables` from the project code are provided on separate pages.

Consider the following (possibly buggy) `test` function and the beginning of the `InsertHashTable` function from HW1.

```
int InsertHashTable(HashTable table,
                    HTKeyValue newkeyvalue,
                    HTKeyValue *oldkeyvalue) {
     // Draw diagram when execution reaches HERE
     // Remainder of function omitted...
     ...
}

int test() {
  HashTable ht = AllocateHashTable(2);
  HTKeyValue kv, oldkv;
  kv.key = 1;
  kv.value = (void*)333;
  int result = InsertHashTable(ht, kv, &oldkv);
  return result;
}
```

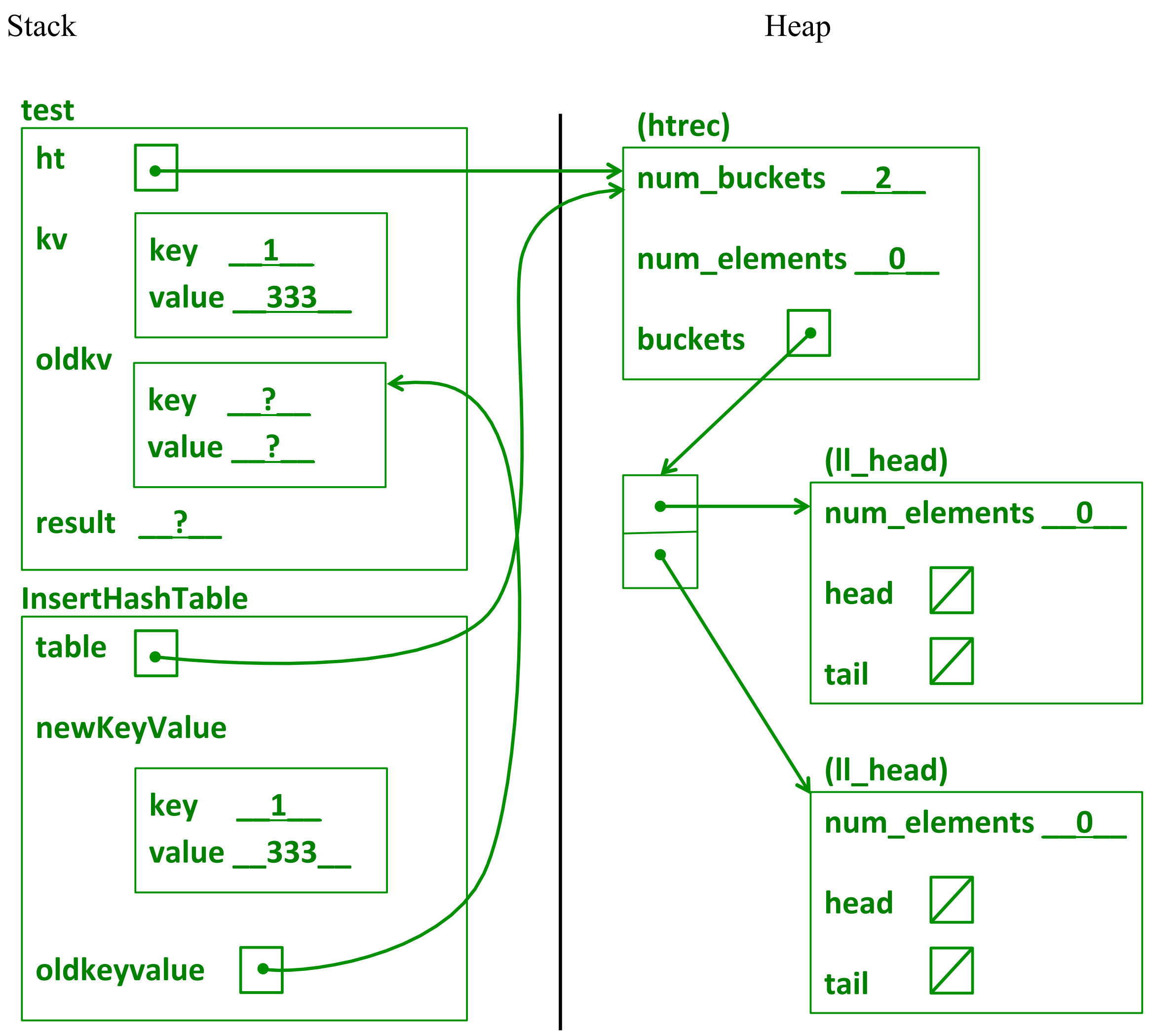
On the next page draw a *precise* diagram of memory at the point when execution reaches the first statement in the body of function `InsertHashTable` (where the "Draw diagram" comment is in the above code).

The data structure definitions involved are given in the `HashTable` and `LinkedList` header files. Also recall the following:

- `AllocateHashTable(`*n*`)` allocates on the heap a new hash table `ht` with an array of *n* buckets. Each bucket element is initialized with the following statement:
  `ht->buckets[i] = AllocateLinkedList();`
  The initialized hash table `ht` is returned as the result of `AllocateHashTable`.
- `AllocateLinkedList()` creates a new `LinkedList` object `ll` on the heap and sets `ll->num_elements = 0` and `ll->head = ll->tail = NULL` before returning `ll` to the caller as the result of `AllocateLinkedList`. No list nodes are allocated when the list is created.

Draw your diagram on the next page. You may remove this page if you wish. **Please do not write on this page – anything written here will be ignored when grading.**

Question 4. (cont.) Draw your diagram below. Local variables and data on the stack should be drawn on the left, and there should be boxes indicating which memory belongs to function `test` and which belongs to function `InsertHashTable`. Draw data allocated on the heap to the right. Use arrows to show how objects are linked by pointers.



Note: This solution shows field names in structs as well as types of the HashTable and LinkedList heap data to make the diagram easier to follow. We did not expect to see all of those details in solutions as long as the diagram clearly showed the memory layout and it was clear what was going on.

Question 5. (18 points) Consider the following C++ class, which implements a simple wrapper around an integer variable, and a main function that uses it. This does compile and execute without errors.

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

class Int {
public:
  Int(): n_(0)           { cout << "default ctr" << endl; }
  Int(int n): n_(n)      { cout << "ctr " << n_ << endl; }
  Int(const Int &other): n_(other.n_)
                         { cout << "copy ctr " << n_ << endl; }
  Int &operator=(const Int & other) {
    cout << "op= " << n_ << "=" << other.n_ << endl;
    if (this == &other) return *this;
    n_ = other.n_;
    return *this;
  }
  ~Int() { cout << "dtr " << n_ << endl; }
private:
  int n_;
};

int main() {
  Int n1 = 1;
  Int n2;
  cout << "---" << endl;
  n2 = n1;
  cout << "---" << endl;
  n1 = 17;
  cout << "---" << endl;
  return EXIT_SUCCESS;
}
```

Answer questions about this code on the next page. You may remove this page if you wish. **Please do not write on this page – anything written here will be ignored when grading.**

Question 5. (cont.) (a) (10 points) What output is printed when this program is executed? (Assume that the compiler generates simple code and does not eliminate any operations or perform similar optimizations on the code.)

```
ctr 1
default ctr
---
op= 0=1
---
ctr 17
op= 1=17
dtr 17
---
dtr 1
dtr 17
```

(b) (8 points) Now suppose that we delete the `&` symbols from the heading of the assignment operator so that it now looks like this:

```
  Int operator=(const Int other) { ... }
```

No other changes are made to the code. Amazingly enough, the program still compiles and executes with no apparent problems. What output is printed when the program is executed after this change? (As with part (a), assume that the compiler is not clever enough to eliminate any operations or otherwise optimize the code.)

```
ctr 1
default ctr
---
copy ctr 1
op= 0=1
copy ctr 1
dtr 1
dtr 1
---
ctr 17
op= 1=17
copy ctr 17
dtr 17
dtr 17
---
dtr 1
dtr 17
```

Question 6. (22 points) Recall the small C++ string class from lecture. `Str.h` contains

```
class Str {
 public:
  Str();                  // default constructor
  Str(const char *s);  // c-string constructor
  Str(const Str &s);   // copy constructor
  ~Str();                 // destructor

  // operations
  int length() const;              // = length of this string
  char * c_str() const;            // = new char* copy of this string
  Str &operator=(const Str &s);   // assignment

  // stream output
  friend std::ostream &operator<<(std::ostream &out, const Str &s);

 private:
  char *st_;  // c-string on heap with data bytes terminated by '\0'
};
```

We would like to create a modified version of `Str`. We've removed the `append` function that was part of the class, and we'd like to add "multiplication" operators. The new `*` operators should work as follows: if `s` is a `Str` object, and `k` is an int, then both `s*k` and `k*s` should create and return a new `Str` object (*not* a pointer to a `Str`) holding `k` copies of `s`. For example, if `s` contains the string "ha", then 3*s should return a `Str` that contains "hahaha". The `*` operator should not modify `s`. In addition there should be a new `*=` assignment operator such that `s*=k` modifies `s` so it now has the value `s*k`. For all of these operators, if `k<=0` then the result should be an empty string with length 0 (i.e., `st_` should point to an array with only a '\0' byte at the end).

(a) (8 points) Write declarations for the new `*` and `*=` operators to be added to the header file `Str.h`. You should show clearly which declarations are added to the class as either member or friend functions, and which declarations are additional overloaded functions that are not part of the class. You should follow best practices in deciding which functions to include in the class and which ones should not be included.

Additional declarations to be added to class `Str` in `Str.h`:

```
Str & operator*=(int k);
```

Additional declarations to add to `Str.h` that are not part of class `Str`:

```
Str operator*(int k, const Str &s);
Str operator*(const Str &s, int k);
```

Note: answers that had the correct function prototypes received most of the credit even if they were declared as `friend` functions or members of the `Str` class when not strictly necessary.

Question 6. (cont) (b) (14 points) Give the code to be added to file `Str.cc` to implement the new `*` and `*=` operators added to `Str.h` on the previous page.

```
// multiplication assignment operator in class Str

Str & Str::operator*=(int k) {
  // strategy: create an array large enough to hold copies,
  // then cat that many copies onto an initial empty string
  int ncopies = k<=0 ? 0 : k;
  char * newst = new char[ncopies*strlen(st_) + 1];
  newst[0] = '\0';
  for (int i = 1; i <= ncopies; i++) {
    strcat(newst, st_);
  }
  delete [] st_;
  st_ = newst;
  return *this;
}

// non-member overloaded multiplication operators

Str operator*(int k, const Str &s) {
  Str result(s);
  result *= k;
  return result;
}
Str operator*(const Str &s, int k) {
  Str result(s);
  result *= k;
  return result;
}
```

Note: Arguably it would be better to use safe functions like `strncat` instead of `strcat`. But since the only data being manipulated is private to the class, any buffer overruns represent a programming error as opposed to bad client data that the class might be expected to defend against.

It might be a little better to have the two `operator+` functions call a common third function to eliminate redundant code, but for an exam question this solution is plenty good enough.

Question 7. (2 free points) (All reasonable answers receive the points. All answers are reasonable as long as there is an answer. ☺)

The traditional last midterm question.

(a) (1 point) What question were you expecting to appear on this exam that wasn't included?

Something different than what was here ☺

(b) (1 point) Should we include that question on the final exam? (circle or fill in)

Yes

No

Heck No!!

$!@$^*% No !!!!!

None of the above. My answer is **Maybe.**