CSE 333 22sp Midterm Exam 5/6/22 Sample Solution

Question 1. (16 points, 4 each part) Building things. Consider these two C files:

a.c

```
void f(char c);

int main(int argc, char**argv){
  f('x');
  return 0;
}
```

b.c

```
void f(int *p) {
  *p = 17;
}
```

(a) Why is the program made from `a.c` and `b.c` incorrect? What would you expect to happen if it is executed?

The declaration of `f` in `a.c` is inconsistent with the actual function definition in `b.c`. Function `f` expects a valid pointer to an integer but it is given the integer value of `x` (which is 120 decimal, but the exact value is irrelevant). It will probably segfault when it attempts to store 17 in that memory location.

(b) Will `gcc -Wall -g -c a.c` and `gcc -Wall -g -c b.c` give an error or will they successfully produce `a.o` and `b.o` without complaint? (Note that `-c` implies that `gcc` will produce an output file named `x.o` from `x.c` if `-o` is not also given.)

Both will compile without errors. (Each file is compiled independently with no knowledge of the internal details of other files.)

(c) Will `gcc -Wall -g -o pgm a.c b.c` give an error or will it successfully produce an executable file `pgm` without complaint?

The code will compile and link with no problems. The linker does not have type information, and since `f` exists in `b.o`, it can be linked with `a.o` to produce an executable program.

(d) How would you use standard C coding practices (using an extra file) to avoid the problems with this program (or at least detect them properly)? Give the contents of the extra file below and explain what modifications should be made to `a.c` and `b.c`, if any.

Create a file `b.h` containing the following:

```
#ifndef B_H_
#define B_H_
void f(int *p);
#endif
```

Add the line

```
#include "b.h"
```

at the beginning of files `a.c` and `b.c`. Remove the declaration

```
void f(char c);
```

from the beginning of `a.c`.

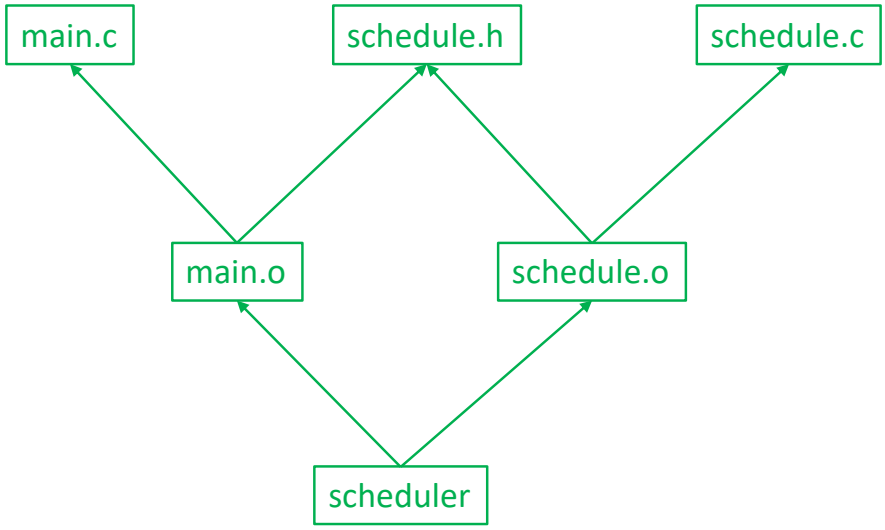
Question 2. (20 points) Making things. The K-Pop group, BTS, is working on building an application for keeping track of their schedule for their upcoming concert. Following good programming practices, they develop the following files:

main.c	schedule.h	schedule.c
`#include "schedule.h"` `...`	`...`	`#include "schedule.h"` `...`

Based on these files, one of the members, Suga, came up with the following commands to build the program based on their understanding of the code:

```
gcc -Wall -std=c17 -g -c main.o main.c schedule.h
gcc -Wall -std=c17 -g -c schedule.o schedule.c schedule.h
gcc -Wall -std=c17 -g -o scheduler main.o schedule.o
```

(a) (6 points) Draw the dependency diagram that these commands attempt to capture (i.e., which files depend on which other files to build the final target program `scheduler`?)



(b) (4 points) What are the technical error(s) in the above `gcc` commands, if any? Or are those commands ok as written?

There are two errors:

- **`.h` (header) files should not be included as input files on the `gcc` command line. They are read by the preprocessor when it encounters a `#include` directive naming the header file.**
- **The commands used to compile the `.c` files to produce `.o` files either need to have a `-o` option in front of the `.o` file names, or else the `.o` file names should be omitted, letting the `-c` option create files with those names automatically.**

(continued on next page)

Question 2. (cont.) (c) (10 points) Write the contents of a `Makefile` for this program that has the following properties:

- The command `make` should build the program `scheduler` from the source files.
- The command `make` should only recompile and link files that need to be rebuilt after any changes to source files. Existing `.o` files and other files should not be re-compiled or relinked if that is not needed to bring the program up to date.
- The command `make clean` should delete the executable file `scheduler` and all of the `.o` files in the current directory.

For reference, here is the table of file names from the previous page:

main.c	schedule.h	schedule.c
`#include "schedule.h"` `...`	`...`	`#include "schedule.h"` `...`

Write the `Makefile` code below.

```
# default target - must be first
scheduler: main.o schedule.o
	gcc -Wall -g -std=c17 -o scheduler main.o schedule.o

# individual source files
schedule.o: schedule.c schedule.h
	gcc -Wall -g -std=c17 -c schedule.c

main.o: main.c schedule.h
	gcc -Wall -g -std=c17 -c main.c

clean:
	rm -rf scheduler *.o
```

Note: solutions that included `-o` options for the `.o` targets (e.g., `-o main.o`) are also fine.

Question 3. (20 points) HW1 linked lists and hash tables revisited. We would like to add a new public function to the HashTables we created in hw1 to retrieve a list of the keys found in the <key,value> pairs in a Hashtable. The specification of this function is:

```
// Return a pointer to a new LinkedList on the heap
// containing all of the keys found in the HashTable table.
// The order in which the keys appear in the resulting list
// is not specified.  The caller is responsible for freeing
// the returned LinkedList when they are finished with it.
LinkedList* HashTable_Keys(HashTable* table);
```

Your code should only use the public interface to the `LinkedList` and `HashTable` modules. Copies of the header files `LinkedList.h` and `HashTable.h` are included at the end of the exam, and you can remove them from the exam if you wish. Your code should allocate any data structures needed, but should not allocate more data than needed and should not have any internal memory leaks. Write your solution below. (Hints: you probably won't need nearly all of this space. The sample solution is about 10-12 lines long, but you don't need to match that. Also, iterators are your friend.)

```
LinkedList* HashTable_Keys(HashTable* table) {
  HTKeyValue_t kv;
  LinkedList* ll = LinkedList_Allocate();
  HTIterator* it = HTIterator_Allocate(table);
  while (HTIterator_IsValid(it)) {
    HTIterator_Get(it, &kv);
    LinkedList_Append(ll, (LLPayload_t) kv.key);
                          // LinkedList_Push is also ok
    HTIterator_Next(it);
  }
  HTIterator_Free(it);
  return ll;
}
```

This solution for appending the key to the list also works, even though it allocates memory that would then have to be freed later.

```
    // alternate append:
    // HTKey_t *key = (HTKey_t *) malloc(sizeof(HTKey_t));
    // *key = kv.key
    // LinkedList_Append(ll, (LLPayload_t) key);
```

During the exam a question was asked about handling errors and we announced that for the exam it was fine to assume that all of the hash table and linked list functions worked and did not return any errors.

Question 4. (20 points) Memory madness. Consider the following program which, as is traditional, does compile successfully, but, contrary to tradition, does not work and, in fact, generates a segfault when it is executed. Even if it didn't crash, it appears to leak memory. Header file `#includes` omitted from the code to save space.

(Corrections for part (b) shown in green bold text)

```
// Capitalize first letter of word
void CapitalizeWord(char* word) {
  word[0] = toupper(word[0]);     // C library function
}

// Return a new c-string containing a copy of word but with
// each character in word duplicated (i.e., if the input ia "abc"
// return "aabbcc")
char* DoubleLetters(char* word) {
  int str_size = strlen(word) + 1;
  char* result = NULL;
  int i = 0;
  int j = 0;
  char c;
  // >>>draw memory diagram
  while (i < str_size) {
    c = word[i];
    result[j] = c;
    result[j + 1] = c;
    i++;
    j++; j+=2;
  }
  result[j] = '\0';
  return result;
}

// main program
int main(int argc, char* argv[]) {
  char cse[] = "cse333";
  printf("1. %s\n", cse);              // expected: "1. cse333"
  CapitalizeWord(cse);
  printf("2. %s\n", cse);              // expected: "2. Cse333"
  char* res_word = DoubleLetters(cse);
  printf("3. %s\n", res_word);     // expected: "3. CCssee333333"
  free(res_word);
  return EXIT_SUCCESS;
}
```

Delete +1 to get correct number of characters in str

Replace `char *result = NULL` with
`char *result = (char*)malloc(2*strlen(word)+1);`
`if (result == NULL) exit (EXIT_FAILURE);`

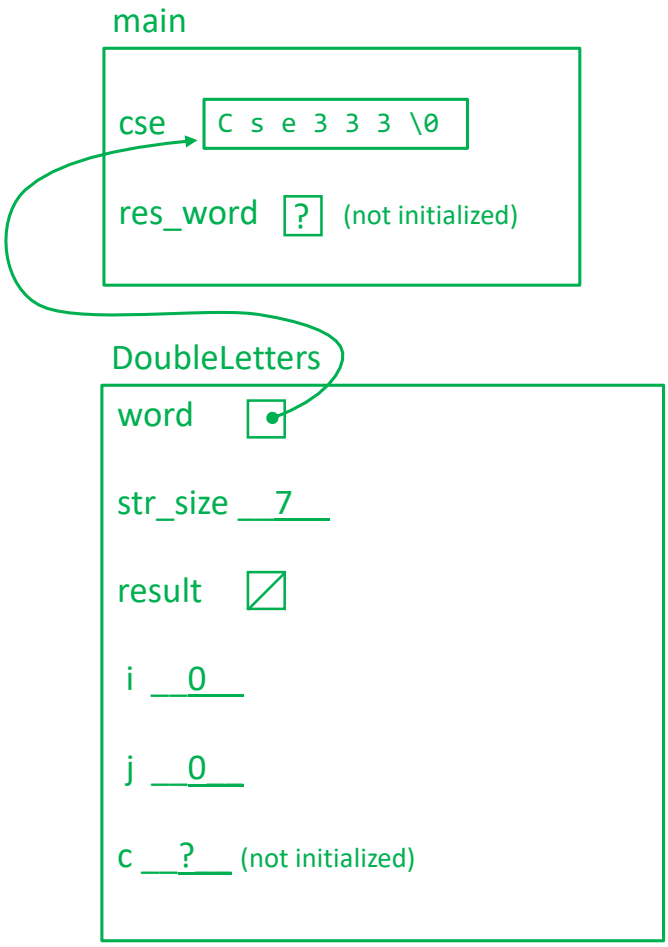
Need to increment j by 2 each time around the loop, then be sure to set final byte of the string to `\0` when done (the `\0` assignment is not needed if `calloc` is used to allocate space).

Need to free `res_word` when done with it

On the next page, (a) draw a memory diagram showing the situation when execution reaches the line marked "HERE" in `DoubleLetters`. Then answer part (b) giving the corrections needed in the code to fix any problems.

(continued on next page)

Question 4. (cont.) (a) (10 points) Draw a boxes 'n arrows diagram showing state of memory when control reaches the comment containing `////HERE////` in the middle of function `DoubleLetters`. Your diagram should have boxes showing the stack frames for all active functions. The stack frames should show values of all local variables. Draw an arrow from each pointer to the location that it references. If there is any data that is allocated on the heap, it 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) (10 points) What is wrong with this program and how should it be fixed? You should identify the bugs and write in the corrections needed on the code listing on the previous page. Cross out anything that should be deleted and add any corrected or new code needed. Be sure it is clear where any additional code or corrections should be inserted.

>>> write your corrections on the code on the previous page <<<

Question 5. (20 points) C++ constructors and things. Here is a fairly small C++ class based loosely on code from sections and from one of our C++ programming exercises. Answer questions about this code on the next page.

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

class Int {
public:
  // constructors and destructors
  Int()            {ival_=1; cout<<"default("<<ival_<<")"<<endl;}
  Int(const int n) {ival_=n; cout<<"ctor("<<ival_<<")"<<endl;}
  Int(const Int& n) {
    ival_ = n.ival_;
    cout << "copyctor(" << ival_ << ")" << endl;
  }
  ~Int() { cout << "dtor(" << ival_ << ")" << endl; }

  // assignment operators
  Int & operator=(const Int & other) {
    if (this != &other) {
      ival_ = other.ival_;
    }
    cout << "assign(" << other.ival_ << ")" << endl;
    return *this;
  }
  Int & operator+=(const Int & other) {
    ival_ += other.ival_;
    cout << "op+=(" << ival_ << ")" << endl;
    return *this;
  }
private:
  int ival_;
};
// additional overloaded operator
Int operator+(const Int & x, const Int & y) {
  cout << "op+" << endl;
  Int sum = x;
  sum += y;
  return sum;
}
```

(continued on next page)

Question 5. (cont.) Now suppose we execute this main program that uses the code from the previous page. What output is produced? (It, and the code from the previous page, compiles and executes successfully)

Hints: remember that variables in a function are initialized (constructed) in declaration order. Destruction order is the reverse of construction order.

```
int main(int argc, char* argv[]) {
  Int a;
  Int b(2);
  Int c = b;
  b = a;
  a = 3+b;
  return EXIT_SUCCESS;
}
```

Output:

```
default(1)
ctor(2)
copyctor(2)
assign(1)
ctor(3)
op+
copyctor(3)
op+=(4)
assign(4)
dtor(4)
dtor(3)
dtor(2)
dtor(1)
dtor(4)
```

Notes: There are two anonymous temporary `Int` objects created and deleted during execution. The exact location and ordering of the destructors could be anywhere after the last use of each temporary and we took that into account while grading.

It is also possible that the compiler could create an extra `Int` temporary for the returned value from `operator+` and then delete it after it is used. Solutions that indicated this also received credit.

Question 6. (4 free points) (All reasonable answers receive the points. All answers are reasonable as long as there is an answer. ☺)

(a) (2 points) What question were you expecting to appear on this exam that wasn't included?

What is the one, true, correct, only way to declare a pointer `p` and initialize it to point to an integer variable `n`?

```
(a)  int *p = &n;
(b)  int* p = &n;
(c)  int * p = &n;
(d)  int*p = &n;
(e)  int *p ☞ n;
```

(b) (2 points) Should we include that question on the final exam? (circle or fill in)

Yes

No

Heck No!!

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

Yes, yes, it *must* be included!!!

No opinion / don't care

None of the above. My answer is _____.