CSE 333 25wi Midterm Exam 2/13/25 Sample Solution

Question 1. (18 points) Making things. We're working with our friend Jungkook on a new music player app. So far, we've got the following files with these `#includes` to reference declarations in various header files:

main.c	playlist.h	playlist.c

`#include "playlist.h"`
`#include "audio.h"`
`...`
	`...`	`#include "playlist.h"`
		`...`

audio.h	audio.c

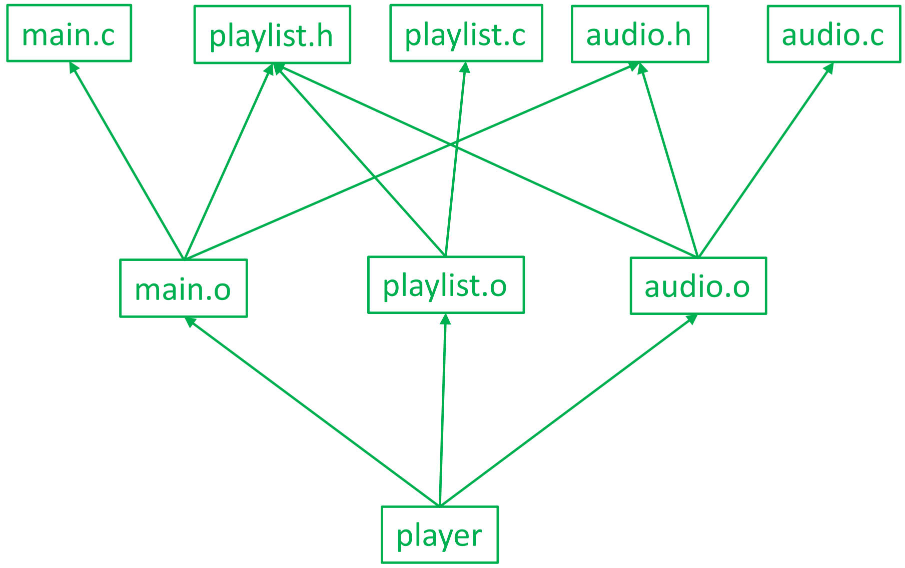
`...`	`#include "playlist.h"`
	`#include "audio.h"`
	`...`

We've been retyping the following `gcc` commands to build the program:

```
gcc -Wall -g --std=c17 -c playlist.c
gcc -Wall -g --std=c17 -c audio.c
gcc -Wall -g --std=c17 -c main.c
gcc -Wall -g --std=c17 -o player audio.o playlist.o main.o
```

Hint: recall that if we compile `foo.c` with the `-c` option and do not specify the output file name (no `-o` option), the output file created will be named `foo.o`, as in the `gcc` commands above.

(a) (6 points) Draw the dependency diagram showing the dependencies between all files used or created during the build process. You should draw an arrow pointing from each file that is built by the `gcc` commands to the file or files that it depends on.



(continued on next page)

Question 1. (cont.) Jungkook has gotten tired of typing `gcc` commands and has written the following `Makefile` to automate compiling and recompiling the `player` program based on the above `gcc` commands, but it doesn't quite work right. It is supposed to build the `player` program by recompiling and relinking only the necessary files.

```
playlist.o: playlist.c playlist.h
	gcc -Wall -g --std=c17 -c playlist.c

audio.o: audio.c audio.h playlist.h
	gcc -Wall -g --std=c17 -c audio.c

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

player: audio.o playlist.o main.o
	gcc -Wall -g --std=c17 -o player audio.o playlist.o main.o
```

Answer the following questions about this `Makefile` and program:

(b) (2 points) Suppose this `Makefile` and all of the source files (`.c` and `.h`), but no compiler or linker output files are in the same directory. What exactly will happen if we run the command `make` in this directory? (Describe the command(s) that are executed and the result(s) they produce.)

make will execute the command

```
gcc -Wall -g --std=c17 -c playlist.c
```

which will compile playlist.c to produce playlist.o and do nothing else.

(c) (6 points) If running `make` as in part (b) above fails to properly build the `player` program, describe exactly what needs to be done to fix the `Makefile` so that it will work properly. You can either explain your changes below or show the changes needed by writing them on the `Makefile` code given above.

There are two corrections needed:

1. **Move the rule for `player` to the front of the `Makefile` so it is the default target, and**
2. **Add the missing header files to the dependences for `audio.o` and `main.o`.**

See corrections above.

(continued on next page)

Question 1. (cont.) Assume that we have now fixed the `Makefile` so that it recompiles only the necessary source files after any changes and successfully builds the `player` program when we use the `make` command.

(d) (2 points) Assuming that we have fixed all the bugs and that all of the source and output files are up-to-date after running a `make` command, what commands are executed if we modify the file `main.c` and then run `make` again?

```
gcc -Wall -g --std=c17 -c main.c
gcc -Wall -g --std=c17 -o player audio.o playlist.o main.o
```

(e) (2 points) Assuming that we have fixed all the bugs and that all of the source and output files are up-to-date after running a `make` command, what commands are executed if we modify the file `audio.h` and then run `make` again?

```
gcc -Wall -g --std=c17 -c audio.c
gcc -Wall -g --std=c17 -c main.c
gcc -Wall -g --std=c17 -o player audio.o playlist.o main.o
```

Note: the `gcc` commands to compile `audio.c` and `main.c` could run in either order, but both run before the command that creates `player`.

Question 2. (16 points) Preprocessor. Suppose we have the following two C source files:

foo.h

```
#ifndef FOO_H_
#define FOO_H_

#define MAGIC 42
#define SPELL MAGIC + MAGIC
#define LIMIT 333
#define while if

int foo(int n);

#endif // FOO_H_
```

foo.c

```
#include "foo.h"

#define WIZARD "Gandalf"

int foo(int k) {
  int ans = MAGIC + SPELL;
  while (ans < LIMIT) {
    ans = ans + MAGIC + WIZARD;
  }
  return ans;
}
```

Show the output produced by the C preprocessor when it processes file `foo.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?) Hint: remember that the preprocessor does string substitution and does not analyze the C code it produces for correctness.

```
int foo(int n);
int foo(int k) {
  int ans = 42 + 42 + 42;
  if (ans < 333) {
    ans = ans + 42 + "Gandalf";
  }
  return ans;
}
```

Question 3. (20 points) Valgrind and memory. This question concerns the following C program `membugs.c` that copies an array of numbers into a linked list on the heap, prints the contents of the list, and then frees the data.

Warning!! Watch your time and do not get bogged down on this question! Only a small number of fixes are needed once you've found the problem(s).

```
#include <stdio.h>     // for printf
#include <stdlib.h>    // for EXIT_SUCCESS

// Node for linked list of integers
typedef struct Node_st {
   int value;
   struct Node_st* next;
} Node;

// Creates a list of nodes on the heap of numbers
// such that the ith node contains values[i].
// Caller is responsible for freeing the nodes.
// Parameters
//   count: the number of elements in the array and
//   number of nodes in the returned list
//   values: array of numbers to be copied to the list
// returns:
//   pointer to first node of new list
Node* MakeList(int count, int values[]);

int main(int argc, char** argv) {
  int values[] = {1, 2, 3, 4};
  int count = 4;

  // create list
  Node* head_ptr = MakeList(count, values);

  // print values in list
  Node* curr_ptr = head_ptr;
  while (curr_ptr != NULL) {
    int value_to_print = curr_ptr->value;
    printf("%i\n", value_to_print);
    curr_ptr = curr_ptr->next;
  }

  // free list contents
  curr_ptr = head_ptr;
  for (int i = 0; i < count; i++) {
    head_ptr = head_ptr->next;
    free(curr_ptr);
    curr_ptr = head_ptr;
  }
  return EXIT_SUCCESS;
}
```

This should be 4, but does not affect valgrind results

insert

(continued on next page)

Question 3 (cont.) Code continued below:

```

Node* MakeList(int count, int values[]) {
  Node* head = (Node*) malloc(sizeof(Node));
  // ** imagine null check **
  Node* curr_node = head;
  for (int i = 0; i < count; i++) {
    curr_node->value = values[i];                       change
    if (i != (count - 1)) {
      curr_node->next = (Node*) malloc(sizeof(Node));
      // ** null check - omitted to save space**
    } else {
        curr_node -> next = null;
    }                                                   insert
    curr_node = curr_node->next;
  }
  return head;
}
```

Question continues on next page. Remainder of this page left blank to be used as needed while working the problem.

Changes to code to fix bugs shown above in bold green, with callout pointers to indicate where changes were made.

There are other ways to fix the problems, and there are undoubtedly cleaner ways to write the code. Any changes that fix the bugs were acceptable.

(continued on next page)

Question 3. (cont.) The program compiles without errors and runs without crashing. However, when we used valgrind to look for memory bugs, it reported some trouble.

Your job for this question is to examine the valgrind report and the code and then show where the bugs are in the code and how to fix them. You should show corrections by crossing out, changing, or adding code in the listing on the previous two pages. There is extra blank space at the bottom of the previous page for you to use if you need additional space, or if you need space to draw diagrams or do other work to figure out the answers.

```
$ valgrind --leak-check=full --track-origins=yes ./membugs
==2432== Memcheck, a memory error detector
==2432== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==2432== Using Valgrind-3.23.0 and LibVEX; rerun with -h for copyright info
==2432== Command: ./membugs
==2432==
1
2
3
==2432== Conditional jump or move depends on uninitialised value(s)
==2432==    at 0x48CB3EB: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2432==    by 0x48C052E: printf (in /usr/lib64/libc.so.6)
==2432==    by 0x4011B3: main (membugs.c:32)
==2432==  Uninitialised value was created by a heap allocation
==2432==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2432==    by 0x40125E: MakeList (membugs.c:52)
==2432==    by 0x401188: main (membugs.c:26)
==2432==
==2432== Use of uninitialised value of size 8
==2432==    at 0x48BF94B: _itoa_word (in /usr/lib64/libc.so.6)
==2432==    by 0x48CAFDB: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2432==    by 0x48C052E: printf (in /usr/lib64/libc.so.6)
==2432==    by 0x4011B3: main (membugs.c:32)
==2432==  Uninitialised value was created by a heap allocation
==2432==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2432==    by 0x40125E: MakeList (membugs.c:52)
==2432==    by 0x401188: main (membugs.c:26)
==2432==
==2432== Conditional jump or move depends on uninitialised value(s)
==2432==    at 0x48BF95C: _itoa_word (in /usr/lib64/libc.so.6)
==2432==    by 0x48CAFDB: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2432==    by 0x48C052E: printf (in /usr/lib64/libc.so.6)
==2432==    by 0x4011B3: main (membugs.c:32)
==2432==  Uninitialised value was created by a heap allocation
==2432==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2432==    by 0x40125E: MakeList (membugs.c:52)
==2432==    by 0x401188: main (membugs.c:26)
==2432==
==2432== Conditional jump or move depends on uninitialised value(s)
==2432==    at 0x48CB8D3: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2432==    by 0x48C052E: printf (in /usr/lib64/libc.so.6)
==2432==    by 0x4011B3: main (membugs.c:32)
==2432==  Uninitialised value was created by a heap allocation
==2432==    at 0x484482F: malloc (vg_replace_malloc.c:446)
```

```
==2432==    by 0x40125E: MakeList (membugs.c:52)
==2432==    by 0x401188: main (membugs.c:26)
==2432==
==2432== Conditional jump or move depends on uninitialised value(s)
==2432==    at 0x48CB0F7: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2432==    by 0x48C052E: printf (in /usr/lib64/libc.so.6)
==2432==    by 0x4011B3: main (membugs.c:32)
==2432==  Uninitialised value was created by a heap allocation
==2432==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2432==    by 0x40125E: MakeList (membugs.c:52)
==2432==    by 0x401188: main (membugs.c:26)
==2432==
0
==2432== Conditional jump or move depends on uninitialised value(s)
==2432==    at 0x4011C5: main (membugs.c:30)
==2432==  Uninitialised value was created by a heap allocation
==2432==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2432==    by 0x40125E: MakeList (membugs.c:52)
==2432==    by 0x401188: main (membugs.c:26)
==2432==
==2432== Conditional jump or move depends on uninitialised value(s)
==2432==    at 0x4847AFF: free (vg_replace_malloc.c:989)
==2432==    by 0x4011E7: main (membugs.c:39)
==2432==  Uninitialised value was created by a heap allocation
==2432==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2432==    by 0x40125E: MakeList (membugs.c:52)
==2432==    by 0x401188: main (membugs.c:26)
==2432==
==2432==
==2432== HEAP SUMMARY:
==2432==     in use at exit: 32 bytes in 2 blocks
==2432==   total heap usage: 5 allocs, 3 frees, 1,088 bytes allocated
==2432==
==2432== 16 bytes in 1 blocks are definitely lost in loss record 1 of 2
==2432==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2432==    by 0x40121B: MakeList (membugs.c:46)
==2432==    by 0x401188: main (membugs.c:26)
==2432==
==2432== 16 bytes in 1 blocks are definitely lost in loss record 2 of 2
==2432==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2432==    by 0x40125E: MakeList (membugs.c:52)
==2432==    by 0x401188: main (membugs.c:26)
==2432==
==2432== LEAK SUMMARY:
==2432==    definitely lost: 32 bytes in 2 blocks
==2432==    indirectly lost: 0 bytes in 0 blocks
==2432==      possibly lost: 0 bytes in 0 blocks
==2432==    still reachable: 0 bytes in 0 blocks
==2432==         suppressed: 0 bytes in 0 blocks
==2432==
==2432== For lists of detected and suppressed errors, rerun with: -s
==2432== ERROR SUMMARY: 9 errors from 9 contexts (suppressed: 0 from 0)
$
```

Question 4. (16 points) The function on this page and the next opens two files, one for reading and one for writing, and copies the contents of the first file to the second. Your job is to complete the code by filling in the blanks lines with the correct POSIX I/O function calls to handle the file operatons (open, close, read, write).

Here is a summary of some key POSIX I/O functions for your reference.

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)

Below is the code you are to complete. You should assume that all necessary header files have been `#included` and you do not need write any other `#includes`.

```
#define BUFFER_SIZE 1024

void copy_file(const char *source_file, const char *dest_file) {

    int source_fd = ______open(source_file, O_RDONLY);________;

    if (source_fd < 0) {

        perror("Error opening source file");

        return;

    }

    int dest_fd = _____open(dest_file, O_WRONLY);________;

    if (dest_fd < 0) {

        perror("Error opening destination file");

        ___close__(source_fd);  // Close source file

        return;

    }
```

(continued on next page)

Question 4. (cont.) Below, complete the rest of the function to copy the files.

```
    char buffer[BUFFER_SIZE];

    ssize_t bytes_read, bytes_written, nbytes;

    // ok to assume that input either works or doesn't, but does
    // not need to be retried if a failure is detected

    while ((bytes_read = ____read__(source_fd,
                                    buffer, BUFFER_SIZE)) > 0) {

        bytes_written = 0;
        while (bytes_written < bytes_read) {
            // fill in output function, buffer location, & length
            // (OK to define extra variables here if that helps)



            nbytes = ___write__(dest_fd,

                       __buffer + bytes written_____ ,

                       ___byres read - bytes written_______);

            if (nbytes < 0) {
              perror("error writing to destination file");
              break;
            }
            bytes_written += nbytes;
        }
    }
    if (bytes_read < 0) {
        perror("Error reading from source file");
    }
    // close files

    close(source_fd);

    close (dest_fd);

}
```

Note: there are other ways to keep track of the buffer position and amount to write. Any correct solution received credit.

Question 5. (20 points) Here is one of those slightly maddening C++ programs with a fairly simple class, which is a wrapper for an integer value, and a small program that uses it. The code compiles and executes with no errors. In the box on the right, write the output produced when it runs.

You should assume that all copy constructors, constructors, and destructors are called as specified by the C++ language and not eliminated by possible compiler optimizations

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

class Int {
public:
  Int(): n_(17)       { cout << "default ctr 17" << 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_;
};

// Return copy of value parameter n
Int cloneval(Int n) {
  return n;
}

// return copy of reference parameter n
Int cloneref(Int &n) {
  return n;
}

int main() {
  Int n1 = 42;
  Int n2 = n1;
  cout << "--1--" << endl;
  Int n3;
  n3 = n1;
  cout << "--2--" << endl;
  n3 = cloneval(n1);
  cout << "--3--" << endl;
  n3 = cloneref(n1);
  cout << "--4--" << endl;
  return EXIT_SUCCESS;
}
```

Write the program output here

```
ctr 42
copy ctr 42
--1--
default ctr 17
op= 17=42
--2--
copy ctr 42
copy ctr 42
op= 42=42
dtr 42
dtr 42
--3--
copy ctr 42
op= 42=42
dtr 42
--4--
dtr 42
dtr 42
dtr 42
```

Note: destructors for anonymous temporaries could appear anywhere after the last use of the temporary. The order shown here is what happened when we ran the program.

Question 6. (8 points) Recall the small C++ string class from lecture. Str.h defines the class as follows:

```
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;          // return length of this STR 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'
};
```

One of our colleagues is experimenting with this class to see if they understand how it works, and they decided to add a *= operator that would "multiply" the contents of a Str by concatenating it with itself the specified number of times. Here is an example:

```
  Str hi("howdy");   // use existing char* constructor to create Str hi
  hi *= 3;
  cout << hi << endl; // writes “howdyhowdyhowdy”
```

(a) (3 points) Give a declaration of the *= operator function as it would be written in class Str in the Str.h header file (i.e., what needs to be added to the above class definition?):

```
   // change s to n copies of s
   Str & operator*=(int n);
```

(b) (5 points) Give the implementation of the *= operator as it would appear in the Str.cc file that implements class Str. (To simplify things, you may assume the operand of *= is an integer > 0. Hint: the C-string operations summarized on the reference page at the end of the exam may be useful here, especially strcat.)

Here is one possible solution:

```
   Str & Str::operator*=(int n) {
     char *newst = new char[strlen(st_)*n +1];
     newst[0] = '\0';
     for (int i = 0; i < n; i++) {
       strcat(newst, st_);
     }
     delete [] st_;
     st_ = newst;
     return *this;
   }
```

Note: We should have allocated more points to this question since it turned out to be a bit more complicated than expected. Because of that, we graded it somewhat like an exercise: 5=perfect, 4=minor fault, etc.

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

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

Implement a clone of emacs. Be sure to include all commands and modes, including an elisp interpreter, a vi mode, and a mode that emulates VSCode for newer programmers. The program must be implemented in C or in assembly language for your choice of processor.

(b) (1 point) 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 ___________________________________.