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
<pre>#include "playlist.h"</pre>	• • •	<pre>#include "playlist.h"</pre>
<pre>#include "audio.h"</pre>		•••

audio.h	audio.c
•••	<pre>#include "playlist.h"</pre>
	<pre>#include "audio.h"</pre>

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.



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.

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	foo.c
#ifndef FOO_H_	#include "foo.h"
#define FOO_H_	Hasting WIGARD UCandalfu
#define MAGIC 42	#deline wizard Gandali
#define SPELL MAGIC + MAGIC	int foo(int k) {
#define LIMIT 333	<pre>int ans = MAGIC + SPELL;</pre>
#define while if	while (ans < LIMIT) {
	ans = ans + MAGIC + WIZARD;
int foo(int n);	}
	return ans;
#endif // FOO_H_	}

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>
 1
                            // for printf
                            // for EXIT SUCCESS
 2
    #include <stdlib.h>
 3
   // Node for linked list of integers
 4
    typedef struct Node st {
 5
 6
       int value;
 7
       struct Node st* next;
 8
   } Node;
 9
10
   // Creates a list of nodes on the heap of numbers
11
    // such that the ith node contains values[i].
    // Caller is responsible for freeing the nodes.
12
13
   // Parameters
    // count: the number of elements in the array and
14
15 // number of nodes in the returned list
16
   // values: array of numbers to be copied to the list
    // returns:
17
18
   // pointer to first node of new list
19 Node* MakeList(int count, int values[]);
20
21 int main(int argc, char** argv) {
                                                   This should be 4, but
22
      int values[] = \{1, 2, 3, 4\};
23
      int count = 4; -
                                                      does not affect
24
                                                     valgrind results
25
      // create list
26
      Node* head ptr = MakeList(count, values);
27
28
      // print values in list
29
      Node* curr ptr = head ptr;
30
      while (curr ptr != NULL) {
31
        int value to print = curr ptr->value;
        printf("%i\n", value_to_print);
32
33
        curr ptr = curr ptr->next;
34
      }
35
                                         insert
36
      // free list contents
      curr ptr = head ptr; .
37
      for (int i = 0; i < \text{count}; i++) {
38
        head ptr = head ptr->next;
39
        free(curr ptr);
40
        curr ptr = head ptr;
41
      }
42
      return EXIT SUCCESS;
43
    }
```

Question 3 (cont.) Code continued below:

```
44
45 Node* MakeList(int count, int values[]) {
46
      Node* head = (Node*) malloc(sizeof(Node));
47
      // ** imagine null check **
48
      Node* curr node = head;
49
      for (int i = 0; i < count; i++) {</pre>
50
        curr node->value = values[i];
                                                     change
        if (i != (count - 1)) { -
51
          curr node->next = (Node*) malloc(sizeof(Node));
52
          // ** null check - omitted to save space**
53
54
        } else {
            curr node -> next = null; -
        }
                                                     insert
55
        curr node = curr node->next;
56
      }
57
      return head;
58
    }
```

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.

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 possibly lost: 0 bytes in 0 blocks ==2432== ==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.

```
(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,
Note: there are other
ways to keep track
                                 buffer + bytes written ,
of the buffer position
and amount to write.
                                  byres read - bytes written );
Any correct solution
                    if (nbytes < 0) {
received credit.
                      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);
```

}

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; }</pre>
  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 ;
                                              Write the program output here
    return *this;
  }
                                                    ctr 42
  ~Int() { cout << "dtr " << n << endl; }
                                                    copy ctr 42
private:
  int n ;
                                                    --1--
};
                                                    default ctr 17
// Return copy of value parameter n
                                                    op= 17=42
Int cloneval(Int n) {
                                                    --2--
 return n;
}
                                                    copy ctr 42
                                                    copy ctr 42
// return copy of reference parameter n
Int cloneref(Int &n) {
                                                    op= 42=42
  return n;
                                                    dtr 42
}
                                                    dtr 42
int main() {
                                                    --3--
  Int n1 = 42;
                                                    copy ctr 42
  Int n^2 = n^1;
                                                    op= 42=42
  cout << "--1--" << endl;
  Int n3;
                                                    dtr 42
  n3 = n1;
                                                    --4--
  cout << "--2--" << endl;
                                                    dtr 42
  n3 = cloneval(n1);
                                                    dtr 42
  cout << "--3--" << endl;
                                                    dtr 42
  n3 = cloneref(n1);
  cout << "--4--" << endl;
                                              Note: destructors for anonymous
  return EXIT SUCCESS;
                                              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:

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"</pre>
```

(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;
}
</pre>
Note: We s
more point
it turned o
complicate
Because of
somewhat
5=perfect,
```

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. O)

(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 _____