Create Rubric

85 points

Create your rubric now or come back to it later. You can also make edits to your rubric while grading.

Q1

9 points

Q1.1 a

1 point

Rubric Settings

○ ○ ○ ○

1 +1.0

Correct: neither an AO nor an ADT should reveal their inner state representation, which would break encapsulation. For an ADT, the representation is hidden by having in the interface only a declaration, not a definition, of the structure whose fields are the state. For an AO, the representation is hidden by making the state variables `static` so that they are not visible outside the file.

2 +0.0

The correct answer was "Neither": neither an AO nor an ADT should reveal their inner state representation, which would break encapsulation. For an ADT, the representation is hidden by having in the interface only a declaration, not a definition, of the structure whose fields are the state. For an AO, the representation is hidden by making the state variables `static` so that they are not visible outside the file.

Add Rubric Item | Create Group | Import...

Q1.2 b

1 point

Rubric Settings

○ ○ ○ ○

1 +1.0

Correct: the opaque pointer allows information hiding, because the interface exposes a declaration not a definition of the state struct and offers a pointer to that struct as the "handle" for clients to indicate which instance they are manipulating. An AO, on the other hand, does not have its state defined in the fields of a structure.

2 +0.0

The correct answer was ADT: the ADT's interface file exposes a declaration (but not a definition) of the state struct (allowing for information hiding), and offers a pointer to that struct as the "handle" for clients to indicate which instance they are manipulating. An AO, however, does not have its state defined in the fields of a structure.

Add Rubric Item	Create Group	Import...

Q1.3 c

1 point

Rubric Settings

○ ○ ○ ○

1 +1.0

Correct: in an AO, there is only ever one instance so there is no need for the clients to specify which instance they are manipulating. In an ADT, though, there may be multiple instances, so the interface functions take an instance pointer as ar argument to specify which instance they are manipulating.

2 +0.0

The correct answer was AO: in an AO, there is only ever one instance so there is no need for the clients to specify which instance they are manipulating. In an ADT, though, there may be multiple instances, so the interface functions take an instance pointer as an argument to specify which instance they are manipulating.

Add Rubric Item | Create Group | Import...

Q1.4 d

1 point

Rubric Settings

○ ○ ○ ○

1 +1.0

Correct: an ADT has a definition of the struct whose field are its state, but typically does not have any global *variables*. An AO, on the other hand, defines its internal state by a set of file-scope static variables.

2 +0.0

The correct answer was AO. An ADT has a definition of the function whose field are its state, but typically does not have any global *variables*. An AO, on the other hand, defines its internal state by a set of file-scope static variables.

Add Rubric Item | Create Group | Import...

Q1.5 e

1 point

⚙ Rubric Settings

○ ○ ○ ○

1 +1.0

Correct: an ADT has a definition of the struct whose fields are its state, so in its constructor it can allocate enough space for one of these structs and return the pointer to the caller. An AO, on the other hand, defines its internal state by a set of file-scope static variables, for which there will always be only one copy of each.

2 +0.0

The correct answer was ADT: an ADT has a definition of the struct whose fields are its state, so in its constructor it can allocate enough space for one of these structs and return the pointer to the caller. An AO, on the other hand, defines its internal state by a set of file-scope static variables, for which there will always be only one copy of each.

✚ Add Rubric Item	Create Group	Import...

Q1.6 f

1 point

⚙ Rubric Settings

○ ○ ○ ○

1 +1.0

Correct: decomment.c did not implement an object at all. It was a program that simulated a DFA to do character processing.

2 +0.0

The correct answer was Neither: decomment.c did not implement an object at all. It was a program that simulated a DFA to do character processing.

✚ Add Rubric Item | Create Group | Import...

Q1.7 g

1 point

Rubric Settings

○ ○ ○ ○

1 +1.0

Correct: the module declared in str.h and implemented in str.c did not implement an object at all. It was a stateless module of functions that were passed strings as arguments but did not define the type or "own" them.

2 +0.0

The correct answer was Neither: the module declared in str.h and implemented in str.c did not implement an object at all. It was a stateless module of functions that were passed strings as arguments but did not define the type or "own" them.

✚ Add Rubric Item | Create Group | Import...

Q1.8 h

1 point

Rubric Settings

○ ○ ○ ○

1 +1.0

Correct: the SymTable module was an ADT with an interface defined by symtable.h and two different implementations, symtablelist.c and symtablehash.c .

2 +0.0

The correct answer was ADT: the SymTable module was an ADT with an interface defined by symtable.h and two different implementations, symtablelist.c and symtablehash.c .

+ Add Rubric Item | Create Group | Import...

Q1.9 i

1 point

Rubric Settings

○ ○ ○ ○

1 +1.0

Correct: in A4 Part 2, dt.c implemented an AO for the directory tree, with one of its state variables being a Node_T , which is an opaque pointer type handle for the ADT defined in nodeDT.c . (DynArray and Path were other ADTs used as part of a DT, although these were shared across the AOs from all three parts of A4.)

2 +1.0

The correct answer is Both: in A4 Part 2, you were right that dt.c implemented an AO for the directory tree, but it did sowith one of its state variables being a Node_T , which is an opaque pointer type handle for the ADT defined in nodeDT.c that was also provided for Part 2. (DynArray and Path were other ADTs used as part of a DT, although these were shared across the AOs from all three parts of A4.) We awarded credit for this answer because we found it plausible that students interpreted "(DT)" as dt.c ".

3 +0.0

The correct answer is Both: in A4 Part 2, dt.c implemented an AO for the directory tree, with one of its state variables being a Node_T , which is an opaque pointer type handle for the ADT defined in nodeDT.c . (DynArray and Path were other ADTs used as part of a DT, although these were shared across the AOs from all three parts of A4.)

Add Rubric Item | Create Group | Import...

Q2

4 points

Rubric Settings

1 +4.0

Correct: (char * (*) (void * , void *))

2 +3.0

The correct answer is: (char * (*) (void * , void *))
Partial credit was awarded for including components of the correct answer.

3 +2.0

The correct answer is: (char * (*) (void * , void *))
Partial credit was awarded for including components of the correct answer.

4 +1.0

The correct answer is: (char * (*) (void * , void *))
Partial credit was awarded for including components of the correct answer.

5 +0.0

The correct answer is: (char * (*) (void * , void *)) .

➕ Add Rubric Item | Create Group | Import...

Q3

13 points

Q3.1 a

1 point

⚙ Rubric Settings

☐ ☐ ☐ ☐ ☐ ☐ ☐

1 +1.0

Correct: two.h is not a target.

2 +0.0

The correct answer was None: two.h is not a target.

➕ Add Rubric Item | Create Group | Import...

Q3.2 b

1 point

Rubric Settings

☐ ☐ ☐ ☐ ☐ ☐ ☐

1 +1.0

Correct: Only #B is executed, because the provided target is specifically `zero.o`.

2 +0.0

The correct answer was only #B, because the provided target is specifically `zero.o`, so the link into the final executable `prog` is not attempted.

Add Rubric Item | Create Group | Import...

Q3.3 c

2 points

Rubric Settings

☐ ☐ ☐ ☐ ☐ ☐ ☐

1 +2.0

Correct: `make` with no arguments will try to build the first target (`prog`). `zero.o` is already up to date after (b.), but the rest of the `.o` files need to be built, then the linker must run to combine them all into the final executable.

2 +1.0

The correct answer was ACDEF: `make` with no arguments will try to build the first target (`prog`). `zero.o` is already up to date after (b.), but the rest of the .o files need to be built, then the link must run to combine them all into the final executable.

3 +0.0

The correct answer was ACDEF: make with no arguments will try to build the first target (prog). zero.o is already up to date after (b.), but the rest of the .o files need to be built, then the linker must run to combine them all into the final executable.

➕ Add Rubric Item | Create Group | Import...

Q3.4 d

1 point

Rubric Settings

☐ ☐ ☐ ☐ ☐ ☐ ☐

1 +1.0

Correct: When we touch zero.c, the C source file is now newer than its corresponding .o file, so when we make, zero.o must be rebuilt, which will make it newer than prog, so prog must be rebuilt as well.

2 +0.0

The correct answer was A and B: When we touch zero.c, the C source file is now newer than its corresponding .o file, so when we make, zero.o must be rebuilt, which will make it newer than prog, so prog must be rebuilt.

➕ Add Rubric Item | Create Group | Import...

Q3.5 e

1 point

⚙ Rubric Settings

☐ ☐ ☐ ☐ ☐ ☐ ☐

1 +1.0

Correct: no files have been updated since we last ran make in d..

2 +0.0

The correct answer was None: no files have been updated since we last ran make in d..

+ Add Rubric Item | Create Group | Import...

Q3.6 f

1 point

⚙ Rubric Settings

☐ ☐ ☐ ☐ ☐ ☐ ☐

1 +1.0

Correct: two.h is a prerequisite for two.o, so when it becomes newer than two.o the object file must be rebuilt. And after doing so, when the object file is newer than the executable, the executable also must be relinked.

2 +0.0

The correct answer was A and D: two.h is a prerequisite for two.o, so when it becomes newer than two.o the object file must be rebuilt. And after doing so, when the object file is newer than the executable, the executable also must be relinked.

✚ Add Rubric Item | Create Group | Import...

Q3.7 g

2 points

Rubric Settings

☐ ☐ ☐ ☐ ☐ ☐ ☐

1 +2.0

Correct: four.h is a prerequisite for {two,three,four}.o , so when it becomes newer than those the object files must be rebuilt. And after doing so, when the object files become newer than the executable, the executable also must be relinked.

2 +1.0

The correct answer was ADEF: four.h is a prerequisite for {two,three,four}.o , so when it becomes newer than those the object files must be rebuilt. And after doing so, when the object files become newer than the executable, the executable also must be relinked.

3 +0.0

The correct answer was ADEF: four.h is a prerequisite for {two,three,four}.o , so when it becomes newer than those the object files must be rebuilt. And after doing so, when the object files become newer than the executable, the executable also must be relinked.

✚ Add Rubric Item | Create Group | Import...

Q3.8 h

1 point

Rubric Settings

☐ ☐ ☐ ☐ ☐ ☐ ☐

1 +1.0

Correct: after touching the .o files, they are still newer than the .c files, so they don't need to be rebuilt, but they are also now newer than the executable, so it does need to be re-linked.

2 +0.0

The correct answer was only A: after touching the .o files, they are still newer than the .c files, so they don't need to be rebuilt, but they are also now newer than the executable, so it does need to be re-linked.

Add Rubric Item | Create Group | Import...

Q3.9 i

1 point

Rubric Settings

☐ ☐ ☐ ☐ ☐ ☐ ☐

1 +1.0

Correct: although `one.o` is removed, the `make` argument specifies one specific target `two.o`, so it will never check if `one.o` needs to be built. `two.o` itself is up-to-date.

2 +0.0

The correct answer was None: although `one.o` is removed, the `make` argument specifies one specific target `two.o`, so it will never check if `one.o` needs to be built. `two.o` itself is up-to-date.

✚ Add Rubric Item | Create Group | Import...

Q3.10 j

1 point

⚙ Rubric Settings

☐ ☐ ☐ ☐ ☐ ☐ ☐

1 +1.0

Correct: after one.o was removed in the previous item but not rebuilt because make was only tasked with rebuilding two.o (if necessary, which it wasn't), one.o is still missing. Thus, this invocation will rectify that and rebuild it, but only one.o (not re-linking prog) since it was the specific target in the make argument.

2 +0.0

The correct answer is only C: after one.o was removed in the previous item but not rebuilt because make was only tasked with rebuilding two.o (if necessary, which it wasn't), one.o is still missing. Thus, this invocation will rectify that and rebuild it, but only one.o (not re-linking prog) since it was the specific target in the make argument.

✚ Add Rubric Item | Create Group | Import...

Q3.11 k

1 point

⚙ Rubric Settings

☐ ☐ ☐ ☐ ☐ ☐ ☐

1 +1.0

Correct: after one.o was rebuilt in the previous item, all the .o files are up to date but we have to relink prog because of the update to one.o .

2 +0.0

The correct answer is only A: after one.o was rebuilt in the previous item, all the .o files are up to date but we have to relink prog because of the update to one.o .

Add Rubric Item | Create Group | Import...

Q4

6 points

Q4.1 a

2 points

Rubric Settings

1 +2.0

Correct: Arrays' elements are stored at memory addresses in increasing order. Thus, starting at the address of auc , the two bytes are 0x01 and 0x03. A short is a multibyte construct, so to interpret this data as a short we need to account for endianness. On a little-endian machine, the byte with the least significant bits is stored first, so that means that 0x01 is the lower order byte and 0x03 is the higher order byte. The value 0x0301 is 769 ($3 * 256 + 1$, or, equivalently in binary 0000001100000001: $= 512 + 256 + 1$). The value should be printed in base 10 as noted in the problem regarding the %hd specifier.

2 +1.0

Arrays' elements are stored at memory addresses in increasing order. Thus, starting at the address of auc , the two bytes are 0x01 and 0x03. A short is a multibyte construct, so to interpret this data as a short we need to account for endianness. On a little-endian machine, the byte with the least significant bits is stored first, so that means that 0x01 is the lower order byte and 0x03 is the higher order byte. The value 0x0301 is 769 ($3 * 256 + 1$, or, equivalently in binary 0000001100000001: $= 512 + 256 + 1$). The value should be printed in base 10 as noted in the problem regarding the %hd specifier.

3 +0.0

Arrays' elements are stored at memory addresses in increasing order. Thus, starting at the address of auc , the two bytes are 0x01 and 0x03. A short is a multibyte construct, so to interpret this data as a short we need to account for endianness. On a little-endian machine, the byte with the least significant bits is stored first, so that means that 0x01 is the lower order byte and 0x03 is the higher order byte. The value 0x0301 is 769 ($3 * 256 + 1$, or, equivalently in binary 0000001100000001: $= 512 + 256 + 1$). The value should be printed in base 10 as noted in the problem regarding the %hd specifier.

Add Rubric Item | Create Group | Import...

Q4.2 b

2 points

Rubric Settings

1 +2.0

Correct: Arrays' elements are stored at memory addresses in increasing order. Thus, starting at the address of auc, the two bytes are 0x01 and 0x03. A short is a multibyte construct, so to interpret this data as a short we need to account for endianness. On a big-endian machine, the byte with the most significant bits is stored first, so that means that 0x03 is the lower order byte and 0x01 is the higher order byte. The value 0x0103 is 259 ($256 + 2 + 1$, or, equivalently in binary 0000000100000011). The value should be printed in base 10 as noted in the problem regarding the %hd specifier.

2 +1.0

Arrays' elements are stored at memory addresses in increasing order. Thus, starting at the address of auc, the two bytes are 0x01 and 0x03. A short is a multibyte construct, so to interpret this data as a short we need to account for endianness. On a big-endian machine, the byte with the most significant bits is stored first, so that means that 0x03 is the lower order byte and 0x01 is the higher order byte. The value 0x0103 is 259 ($256 + 2 + 1$, or, equivalently in binary 0000000100000011). The value should be printed in base 10 as noted in the problem regarding the %hd specifier.

3 +0.0

Arrays' elements are stored at memory addresses in increasing order. Thus, starting at the address of auc, the two bytes are 0x01 and 0x03. A short is a multibyte construct, so to interpret this data as a short we need to account for endianness. On a big-endian machine, the byte with the most significant bits is stored first, so that means that 0x03 is the lower order byte and 0x01 is the higher order byte. The value 0x0103 is 259 ($256 + 2 + 1$, or, equivalently in binary 0000000100000011). The value should be printed in base 10 as noted in the problem regarding the %hd specifier.

Add Rubric Item	Create Group	Import...

Q4.3 c

2 points

Rubric Settings

1 +2.0

Correct: Arrays' elements are stored at memory addresses in increasing order. Thus, if we point s at the same thing auc points to and read 1 byte, that's identical to reading auc[0] , which is 1. The %hd specifier was noted in the problem to print this value in base 10.

2 +1.0

1 is correct, but this answer gives it in the wrong format: Arrays' elements are stored at memory addresses in increasing order. Thus, if we point s at the same thing auc points to and read 1 byte, that's identical to reading auc[0] , which is 1. The %hd specifier was noted in the problem to print this value in base 10.

3 +0.0

The correct answer is 1: Arrays' elements are stored at memory addresses in increasing order. Thus, if we point s at the same thing auc points to and read 1 byte, that's identical to reading auc[0] , which is 1. The %hd specifier was noted in the problem to print this value in base 10.

Add Rubric Item | Create Group | Import...

Q5

8 points

Rubric Settings

0x

1 +1.0

1 point was earned having demonstrated calculating the offset in a procedurally correct way, but with an arithmetic error (typically producing 0x3090 instead of 0x30F0 as though the subtraction was done in decimal rather than hex, or 0x3000 by borrowing incorrectly)

2 +2.0

2 points were earned for: algorithms we could not recognize but that produced an instruction with the given opcode bits or the correct register, or having demonstrated calculating the correct offset or the correct offset except with the order of subtraction reversed.

3 +3.0

3 points were earned for: incorrect algorithms we could recognize that were then compounded with additional mistakes in execution, or algorithms we could not recognize but that produced an instruction with the given opcode bits and the correct register.

4 +4.0

4 points were earned for an incorrect algorithm (>>3 perhaps trying to get to 21 bits from the 24 bits inferred from the result of a 6-hexit subtraction) that was subequently executed perfectly.

5 +5.0

5 points were earned for using the correct algorithm executed perfectly (possibly after incorrect hex subtraction), but flipping the bits or bytes of the answer in an unnecessary and incorrect way.

6 +6.0

6 points were earned for an incorrect algorithm (>>2 like with b , which is wrong because unlike with b we don't know that the target of an ADR is an instruction and thus 4-byte aligned) that was subsequently executed perfectly.

7 +7.0

7 points were earned for a correct algorithm that was executed perfectly ... after having done incorrect hex subtraction to compute the displacement, either by subtracting in the wrong order or making an arithmetic error. (The correct displacement calculation is: 0x217217 - 0x214127 = 0x30F0)

8 +8.0

Correct!

9 +0.0

No credit earned. We could not determine that this response encoded the 0ii1 0000 in the first 8 bits, nor encoded the register in the rightmost five bits, nor calculated the correct offset.

Add Rubric Item | Create Group | Import...

Q6

18 points

Q6.1 a-i

1 point

Rubric Settings

○ ○ ○

1 +1.0

Correct: this is a function declaration, just like we would normally put in our header file. It tells the compiler that there is a function with that name and signature, but does not define (give the implementation for) the function. The definition for this function appears in `file2.c`.

2 +0.0

The correct answer is Other Declaration: this is a function declaration, just like we would normally put in our header file. It tells the compiler that there is a function with that name and signature, but does not define (give the implementation for) the function. The definition for this function appears in `file2.c`.

+ Add Rubric Item | Create Group | Import...

Q6.2 a-ii

1 point

Rubric Settings

○ ○ ○

1 +1.0

Correct: this is the definition of `file1.c`'s internally-linked (because of the `static` keyword) file-scope variable `i`. It will be placed in the BSS section, where it will assume the value 0 at runtime.

2 +0.0

The correct answer is Definition: this is the definition of `file1.c`'s internally-linked (because of the `static` keyword) file-scope

variable i . It will be placed in the BSS section, where it will assu
the value 0 at runtime.

+ Add Rubric Item | Create Group | Import...

Q6.3 a-iii

1 point

Rubric Settings

○ ○ ○

1 +1.0

Correct: this isn't a declaration at all, it's a **function call**.

2 +0.0

The correct answer was "Neither or Error": this isn't a declaration at all, it's a **function call**

+ Add Rubric Item | Create Group | Import...

Q6.4 a-iv

1 point

Rubric Settings

○ ○ ○

1 +1.0

Correct: this is a declaration, but not a definition. The extern keyword on a global variable indicates to the compiler that there exists (somwhere) an externally linked definition for this variable name, so that the compiler can generate assembly language code from the C statements using that variable. The definition for this variable is at Line 7 in file3.c .

2 +0.0

The correct answer is Other Declaration: this is a declaration, but not a definition. The extern keyword on a global variable indicates to the compiler that there exists (somwhere) an externally linked definition for this variable name, so that the compiler can generate assembly language code from the C statements using that variable. The definition for this variable is at Line 7 in file3.c .

+ Add Rubric Item | Create Group | Import...

Q6.5 a-v

1 point

Rubric Settings

○ ○ ○

1 +1.0

Correct: this is the definition of the externally-linked variable i that is used in files 2 and 3. (It is shadowed by the internally-linked i in file1.c)

2 +0.0

The correct answer is Definition: this is the definition of the externally-linked variable i that is used in files 2 and 3. (It is shadowed by the internally-linked i in file1.c)

+ Add Rubric Item | Create Group | Import...

Q6.6 a-vi

1 point

Rubric Settings

○ ○ ○

1 +1.0

Correct: this is a declaration, but not a definition. The extern keyword on a global variable indicates to the compiler that there exists (**somwhere**, *not* necessarily in another file!) an externally linked definition for this variable name. In this case, that definition exists ... on the previous line.

2 +0.0

The correct answer is Other Declaration: this is a declaration, but not a definition. The extern keyword on a global variable indicates to the compiler that there exists (**somwhere**, *not* necessarily in another file!) an externally linked definition for this variable name. In this case, that definition exists ... on the previous line.

Add Rubric Item | Create Group | Import...

Q6.7 b

1 point

Rubric Settings

1 +1.0

Correct: static int i is a file-scope variable, thus it has process duration, and thus it goes in BSS if not given an explicit initialization.

2 +0.0

The correct answer was the BSS section: static int i is a file-scope variable, thus it has process duration, and thus it goes in BSS if not given an explicit initialization.

Add Rubric Item | Create Group | Import...

Q6.8 c

1 point

Rubric Settings

1 +1.0

Correct: on line 5, the declaration indicates to the compiler that there exists an externally linked definition of i in some file. The definition is the one at line 7 in file 3.h, which is a a file-scope variable, thus it has process duration, and thus it goes in DATA because it is given an explicit initialization in the definition.

2 +0.0

The correct answer is the DATA section: on line 5, the declaration indicates to the compiler that there exists an externally linked definition of i in some file. The definition is the one at line 7 in file 3.h, which is a a file-scope variable, thus it has process duration, and thus it goes in DATA because it is given an explicit initialization in the definition.

Add Rubric Item | Create Group | Import...

Q6.9 d

1 point

Rubric Settings

1 +1.0

Correct: Line 7 is a definition of a file-scope variable, thus it has process duration, and thus it goes in DATA because it is given an explicit initialization in the definition.

2 +0.0

The correct answer is the Data section: Line 7 is a definition of a file-scope variable, thus it has process duration, and thus it goes in DATA because it is given an explicit initialization in the definition.

Add Rubric Item | Create Group | Import...

Q6.10 e

1 point

Rubric Settings

1 +1.0

Correct: In `file1.c` the variable `i` is that file's internally-linked one from line 2. Because it is a Process Duration variable that was not initialized, it is placed in the BSS section and assumes the value **0** at runtime. Uninitialized *Temporary Duration* variables are the ones that keep whatever previous value existed in memory.

2 +0.0

The correct answer is 0: In `file1.c` the variable `i` is that file's internally-linked one from line 2, which shadows the externally linked `i` defined in `file3.c`. Because it is a Process Duration variable that was not initialized, it is placed in the BSS section and assumes the value **0** at runtime. Uninitialized *Temporary*

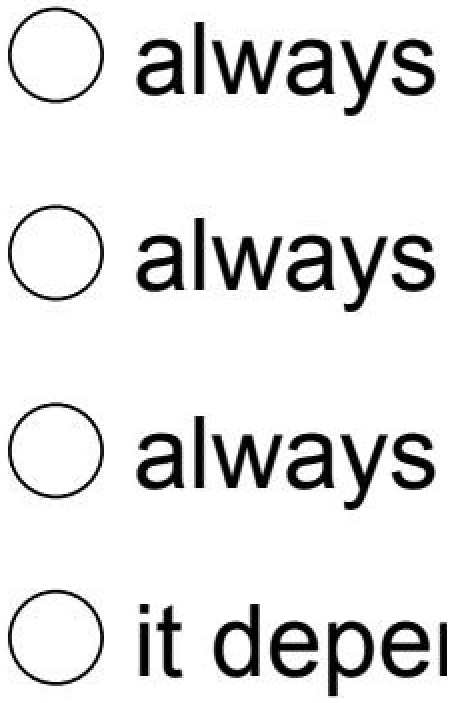
Duration variables are the ones that keep whatever previous valu existed in memory.

Add Rubric Item | Create Group | Import...

Q6.11 f

1 point

Rubric Settings



○ always

○ always

○ always

○ it depe

1 +1.0

Correct: in file2.c the variable i is the one **declared** on Line 5, but that variable is the one **defined** on line 7.

2 +0.0

The correct answer is "always the one defined in file3.c": in file2.c the variable i is the one **declared** on Line 5, but that variable is the one **defined** on line 7. The context of where the function was called from makes no difference on the variable appearing in the function body (unlike, say, if i had been a parameter of the function)

Add Rubric Item | Create Group | Import...

Q6.12 g-i

1 point

Rubric Settings

☐ ☐ ☐ ☐

1 +1.0

Correct: the function declaration shows the **compiler** the function signature for printI so that the compiler can typecheck any function calls of that function and generate the appropriate assembly instructions to prepare arguments and make the function call.

2 +0.0

The function declaration shows the **compiler** the function signature for printI so that the compiler can typecheck any function calls of that function and generate the appropriate assembly instructions to prepare arguments and make the function call.

Add Rubric Item | Create Group | Import...

Q6.13 g-ii

1 point

Rubric Settings

☐ ☐ ☐ ☐

1 +1.0

Correct: the compiler translates from C to assembly, adding labels for global variables, functions, and branch targets.

2 +0.0

The **compiler** translates from C to assembly, adding labels for global variables, functions, and branch targets.

Add Rubric Item | Create Group | Import...

Q6.14 g-iii

1 point

Rubric Settings

☐ ☐ ☐ ☐

1 +1.0

Correct: the **assembler** translates from assembly language to machine language, replacing the compiler's labels with memory section offsets (relative to its own file only).

2 +0.0

The **assembler** translates from assembly language to machine language, replacing the compiler's labels with memory section offsets (relative to its own file only).

Add Rubric Item | Create Group | Import...

Q6.15 g-iv

1 point

Rubric Settings

☐ ☐ ☐ ☐

1 +1.0

Correct: the **preprocessor** handles \#include directives, which take declarations from header files and prepend them to the C code in order to give the compiler the information about the declared functions.

2 +0.0

The **preprocessor** handles \#include directives, which take declarations from header files and prepend them to the C code in order to give the compiler the information about the declared functions. (The linker adds function definitions, i.e., machine code of their implementation, but not declarations to be used earlier in the build process.)

Add Rubric Item | Create Group | Import...

Q6.16 g-v

1 point

Rubric Settings

☐ ☐ ☐ ☐

1 +1.0

Correct: similar to part g.i of this question, the function declaration shows the **compiler** the type of i so that the compiler can generate the appropriate assembly instructions to manipulate it. Without this line, the compiler would interpret i to be an undeclared (presumably local) variable within printI.

2 +0.0

The correct answer is the **compiler**. Similar to part g.i of this question, the function declaration shows the compiler the type of i so that the compiler can generate the appropriate assembly instructions to manipulate it. Without this line, the compiler would interpret i to be an undeclared (presumably local) variable within printI.

Add Rubric Item | Create Group | Import...

Q6.17 g-vi

2 points

Rubric Settings

☐ ☐ ☐ ☐

1 +2.0

Correct: The **assembler** can produce a finalized machine code instruction for a branch within a single file's TEXT section: the difference between two offsets in the same TEXT section will be the same as the difference between their two final addresses. The assembler cannot (and thus will produce a relocation record for the **linker** to patch and finalize) when the branch is to a location defined in another file (e.g. a function call from a client to a module or to the standard library).

2 +1.0

The **assembler** can produce a finalized machine code instruction for a branch within a single file's TEXT section: the difference between two offsets in the same TEXT section will be the same as the difference between their two final addresses. The assembler cannot (and thus will produce a relocation record for the **linker** to patch and finalize) when the branch is to a location defined in another file (e.g. a function call from a client to a module or to the standard library).

3 +0.0

The **assembler** can produce a finalized machine code instruction for a branch within a single file's TEXT section: the difference between two offsets in the same TEXT section will be the same as the difference between their two final addresses. The assembler cannot (and thus will produce a relocation record for the **linker** to patch and finalize) when the branch is to a location defined in another file (e.g. a function call from a client to a module or to the standard library).

Add Rubric Item | Create Group | Import...

Q7

12 points

Q7.1 first four

4 points

⚙ Rubric Settings

1 +4.0

BDGI was the intended answer for this problem, but pulling the FT_getFileContents call out of the same conditional structure (BGID and DBGI) also works.

2 +3.0

This answer had the right four lines, but their order doesn't work. (The most common instance of this is BDIG: this puts an else clause in the middle of the block, which is syntactically incorrect.)

3 +3.0

This answer had three of the four lines correct and in the right place in the sequence, but the fourth line was incorrect.

4 +2.0

This answer had two correct pieces and two incorrect pieces (e.g. wrong line or lines in wrong order)

5 +1.0

1 point of partial credit earned.

6 +0.0

No points earned.

Add Rubric Item | Create Group | Import...

Q7.2 last eight

8 points

Rubric Settings

1 +8.0

Correct: ACEJGFHK is the correct sequence to loop over the indices, checking at each index if there is a mismatch (in which case, print the error message and return FALSE) , and returning true if we reach the end of both strings with a match.

2 +6.0

ACEJGFHK is the correct sequence to loop over the indices, checking at each index if there is a mismatch (in which case, print the error message and return FALSE) , and returning true if we reach the end of both strings with a match.

3 +5.0

ACEJGFHK is the correct sequence to loop over the indices, checking at each index if there is a mismatch (in which case, print the error message and return FALSE) , and returning true if we reach the end of both strings with a match.

4 +3.0

ACEJGFHK is the correct sequence to loop over the indices, checking at each index if there is a mismatch (in which case, print the error message and return FALSE) , and returning true if we reach the end of both strings with a match.

5 +0.0

ACEJGFHK is the correct sequence to loop over the indices, checking at each index if there is a mismatch (in which case, print the error message and return FALSE) , and returning true if we reach the end of both strings with a match.

Add Rubric Item | Create Group | Import...

Q8

15 points

Q8.1 a

5 points

Rubric Settings

1 +5.0

Correct

2 +1.0

Flip conditional correctly

3 +1.0

Goto "else clause" (skip "then clause")

4 +1.0

Goto at end of "then clause" to "After" label (skip "else clause") + "After" label

5 +1.0

"else clause" label

6 +1.0

Statements unchanged from C code

7 +0.0

Answer Left Blank

Add Rubric Item | Create Group | Import...

Q8.2 b

2 points

Rubric Settings

1 +2.0

Correct

2 +1.0

1 point of partial credit was earned.

3 +0.0
Incorrect

+ Add Rubric Item | Create Group | Import...

Q8.3 c

2 points

Rubric Settings

1 +2.0
Correct

2 +1.0
1 point of partial credit was earned.

3 +0.0
Incorrect

+ Add Rubric Item | Create Group | Import...

Q8.4 d

2 points

Rubric Settings

1 +2.0

Correct

2 +1.0

1 point of partial credit was earned.

3 +0.0

Incorrect

Add Rubric Item	Create Group	Import...

Q8.5 e

2 points

Rubric Settings

1 +2.0

Correct line and correct reason

2 +1.0

1 point of partial credit was earned for the correct line but wrong reason or no reason

3 +0.0

Incorrect

+ Add Rubric Item | Create Group | Import...

Q8.6 f

2 points

Rubric Settings

1 +2.0
Correct

2 +1.0
1 point of partial credit was earned.

3 +0.0
Incorrect

+ Add Rubric Item | Create Group | Import...