1. What are the primary responsibilities of the linker? Describe each one in detail.

A linker is a computer program that takes one or more object files generated by a compiler and combines them into one, executable program. It will be responsible for symbol resolution that is to associate each symbol reference with a single symbol definition. It will also be responsible for searching installed libraries to find function code. When the linker has constructed the list of libraries, it repeatedly scans each library in the list to resolve references. Then linker will relocate code and collect all the pieces into a single executable.

2. Provide an in-depth description of each of the following types of files that are produced at different points in the compilation process (including their contents) using the 'gcc'
compiler:

1. .c (source file)

c files are the source files. They typically contain the source code implementation of the functions that were prototyped in the appropriate header file.

2. .s (assembly file)

It's generated by the compiler. S files are assembly language files. They are a form of machine code. It is at a low level of programming. It is automatically generated by the compiler upon successful compilation of every

3. .o (object file)

It's generated by the assembler. It contains compiled object code. Object files will passed to and processed by the linker to produce the final executable.

4. .so (shared object file)

It's generated by grouping assembly code together. Shared object files are linked at run-time by the OS.

3. Why is a .o file not executable even though it contains binary machine code?

Created by assembler, object files can be separate modules and they will contain unsolved references since each source file generally references some things defined in other source files. Assembler doesn't know addresses of external objects when assembling files separately. Assembler doesn't know where the things it's assembling will go in memory. Besides binary machine code, object files also contain the information required to execute it such as debugging information, symbols, stack information, relocation, and profiling information.

4.What information is stored in a relocation record and a symbol table? What is this information used for by the linker?

A relocation record contains information about addresses referenced in the object file that the linker must adjust once it knows the final memory allocation.
Symbol table contains name and current location of variable or procedure that can potentially be referenced in other object files.

The linker reads in all symbols, create complete symbol table in memory during pass 1. Then the linker reads in section and relocation information, update addresses, write out new file during pass 2.

5.Why is the linking step necessary? In other words, why can’t the compiler also perform the job of the linker?

As I’ve mentioned in Q3, the information in object file is incomplete, since each source file generally references some things defined in other source files. A linker is needed to combine all of the object files for one program into a single object file and to re-organize storage allocation.

Executable File Header			
	Text Size	4 0 0	# total combined text size
	Data Size	8 0	# total combined data size
	Address		
Text Segment	0 0 4 0 0 0 0 0_{16} (the memory address for Procedure A's lw instruction)	lw $a0, **X**($gp) X = 8 0 0 0_{16}	# where X is the offset relative to the global pointer register to get to X in the data segment
	0 0 4 0 0 0 0 4_{16} (the memory address for Procedure A's jal instruction)	jal **B** 0 0 4 0 0 3 0 0_{16}	# where B is the (pseudo-direct) address of Procedure B in the text segment
	...		
	0040 0300_{16} (the memory address for Procedure B's lw instruction)	sw $a1, **Y**($gp) Y = 7 9 4 0_{16}	# where Y is the offset relative to the global pointer register to get to Y in the data segment
	0040 0304_{16} (the memory address for Procedure B's jal instruction)	jal **A** 0 0 4 0 0 0 0 0_{16}	# where A is the (pseudo-direct) address of Procedure A in the text segment
	...		
Data Segment	Address		
	1 0 0 0 0 0 0 0_{16}	X	# the location of X in the data section
	1 0 0 0 0 0 6 0_{16}	Y	# the location of Y in the data section

Part 3 - Error Types

For each of the C code snippets below, tell which of the following errors exist: scanner, syntax, semantic, linker, or if there is no error. Note: some snippets may contain multiple types of errors. Please be sure to list all types of warning/errors you find. You might find it helpful to compile your code to identify any warnings/errors: https://repl.it/languages/c

C Code	Error(s)

`#include <stdio.h>`
 	
`int main()`
`{`
`  printf("Today is the $d’th day of April.", day);`
`}`
	Semantic error: variable day was not initialized.
	
	Syntax error:$d should be %d.

`int main(void) {`
`int -i = 0;`
`}`
	Syntax error: can’t have variable name begin with -.
	Linker error: should include the header <stdio.h>

`#include <stdio.h>`
 	
`int main(void) {`
`  int i = -0;`
`  printf("i = %f", i)`
`}`
	Syntax error: should have semicolon after every statement.
	
	Semantic error: i is declared to be int. But %f is for double type.

`#include <stdio.h>`
 	
`int main(void) {`
`  bool loop = true;`
 	
`  while(loop)`
`    printf("Hello, world!\n");`
`  }`
`}`
	Linker error: cant find reference to bool identifier, should include #include <stdbool.h> in the code
	
	Syntax error: missing a closing brace ('}')

`#include <string.h>`
 	
`int main(void) {`
`  do until(i == 10)`
`  {`
`    printf("Hello, world!\n");`
`    i++;`
`  }`
`}`
	Semantic error:
	variable i was not initialized.
	function 'until' was not declared
	
	Linker error:
	should include the header <stdio.h> to use printf

```c
#include <stdio.h>

int main(void) {
  int i = 0;

  do
  {
    printf("Hello, world!\n");
    ++i;
  } while(i == 10)
}
```

Syntax error: expected ';' after do/while statement