Short Questions (20 points)

1. (2 points) What is an automatic variable?
2. (2 points) What type should a function have if it returns something like 2018?
3. (2 points) How do you set a breakpoint at the main function in gdb?
4. (2 points) Give an example of I/O redirection by writing the output of a program named prog to a file named outfile.
5. (6 points) For each pair of values, explain the difference between the ***values***, if there is one. If they are the same, briefly explain why.
	* '\x90' and 0x90
	* '\x10' and 16
	* i++ and ++i
6. (6 points) Write the binary representation of 0x1337d00d.
Then, give its Two's Complement in hex. Then, give its Two's Complement in binary.

Binary : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Two's Complement

Hex : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Binary : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Evaluation Questions (10 points)

1. (6 points) What are the values of a, b, and c from the main function?

#include <stdio.h>

void some\_func(int a, int b, int \*c) {
 a ^= 5;
 \*c = a + b;
 b <<= 4;
}

int main() {
 int a = 3, b = 0x011, c = 0;
 some\_func(a, b, &c);
 a = (b > c) ? b : c;
}

a: \_\_\_\_\_\_\_\_ b: \_\_\_\_\_\_\_\_\_ c: \_\_\_\_\_\_\_\_\_
2. (4 points) What are the values of a and b?

int a = 0xd00d;
char b = (char) 128;
b = b + a;
a = a + '\xdd';

a (rightmost 4 digits, in hex): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b (in hex):
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Memory Allocation Question (10 points)

Indicate how many bytes are ***allocated*** by each line of code, as well as initial values of variables on that line. For multiple values, you can give one answer for all, or comma-separate. For junk, put JUNK.

 Bytes Values
enum bool {FALSE, TRUE}; \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

int a[2]; \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

void main() {

 static char array[6]; \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

 const int b = 7; \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

 {

 enum bool a, b = TRUE; \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

 }

 printf("Hello Student.\n"); \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

}

Error Finding (20 points)
The following code is riddled with syntax and logic errors. For each mistake you find, indicate the lines on which the mistake occurred, what the mistake was, and why it was a mistake. There may be multiple mistakes on one line. Identical mistakes on the same line should be counted as one. There may be more or less errors than mistake slots. Full score is awarded for finding ***enough*** of the mistakes. Draw a smiley face next to this sentence to show that you read all of this.

Line # Code

0000 /\* This program finds the two

0001 smallest numbers in an array \*/

0002 #include <stdio.h>

0003 #define ARRAY\_SIZE = 4

0004 int arr[ARRAY SIZE]={-1,4,5,6};

0005 int FST\_sml, SND\_sml;

0006

0007 void main() {

0008 int i;

0009

0010 for (i=0; i<ARRAY\_SIZE;i--);

0011 if (arr[i] < fst\_sml)

0012 fst\_sml = arr[i];

0013 snd\_sml = fst\_sml;

0014 else if (arr[j]<snd\_sml)

0015 snd\_sml = arr[i]

0016

0017 printf('1st smallest: %s\n', fst\_sml);

0018 printf("2nd smallest: %s\n", snd\_sml);

0019 return "OK";

0020 }

Error Lines What & Why

1. \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8. \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9. \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10. \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program on Paper (40 points)

Write **pseudocode** and **C code** for the following program.

Specifications:

1. The program takes as standard input lines of text.
2. For each line, the program looks for an instance of the word “Boston” (case-sensitive).
3. The program should keep track of how many times it has found the word Boston.
4. The program should also keep track of how many words it has found.
5. When EOF is reached, it should print the counts of how many words were “Boston” and how many words total.

Words are any non-whitespace sequence of characters either starting at the beginning of a string or between two whitespace characters.

Whitespace characters to worry about: ‘\t’ for tab, ‘\n’ for newline, ‘ ’ for space.

Don’t use any I/O or library functions other than the ones described in class or homework.

Examples of acceptable functions: fgets, fprintf, fscanf, getchar, putchar

If you would like to make helper functions for simple tasks, feel free.

**Answer Key**

Short Questions

1. An automatic variable is a local variable that is automatically put on stack memory when needed and taken off stack memory when the function returns/finishes.
2. The function should return int because 2018 is too big to be a char, and it’s an integer value.
3. (gdb) break main
The “(gdb)” bit isn’t part of the command, but it’s just there to indicate that it’s done in the gdb prompt.
4. prog > outfile
	1. The char value on the left ‘\x90’ is a negative number and the int value on the right 0x90 is a positive number. You can tell by checking the sign bit, the leftmost bit of each number if you’re not convinced.
	2. These numbers are the same, ‘\x10’ and 16. Since both numbers fit in the right-most 7 bits, there’s no change in sign. The 10’s digit in hexadecimal is for 16, so 1(16) + 0(1) = 16.
	3. While both values refer to the variable i, we can safely say that i++ and ++i differ by 1. This is because i++ takes the value that i currently is and ++i takes the value 1 higher.
	4. 00010011001101111101000000001101
	5. ecc82ff2
	6. 11101100110010000010111111110010

Evaluation Questions

* 1. 0x17 or 23
	2. 0x11 or 17
	3. 0x17 or 23
1. a
	1. cfea
	2. 8d

Memory Allocation

1. 0 bytes allocated; N/A values
2. 8 bytes allocated; 0, 0
3. 6 bytes allocated; 0, 0, 0, 0, 0, 0 OR 0’s
If it’s clear what you mean, I don’t care how you write it
4. 4 bytes allocated; 7
5. 8 bytes allocated; JUNK, TRUE OR JUNK, 1
6. 0 bytes allocated; N/A values

Error Finding

Line 3, include with equal sign.
Line 4, typo: missing \_ from constant name
Line 5, no initial values: contain junk for comparison on lines 11 and 14
Line 10, i-- makes the loop iterate in the wrong direction
Line 11, wrong case for variable name.
Line 13, didn’t use a block for then-statement so this line hands.
Line 13, logic error: should do this before line 12, otherwise value in fst\_sml is obliterated
Line 14, typo: j instead of i
Line 15, missing semicolon
Line 17, using single quotes for string literal
Line 19, returning from void

Program on Paper

Pseudocode

function ws(c)

Returns true if and only if c is a whitespace character

End function

Main

Skip over all the whitespace at the beginning

Until EoF

 Check the current character

 If I find the beginning of Boston (“B”)

 Look for the rest of the letters.

 If I find all of them

 Increment number of “Boston”

 Else

 Stop where I am and check cases again.

 Otherwise, if I find the beginning of whitespace

 Skip over all the whitespace

 Since I hit whitespace, that means I was at a word before

 So, increment number of Words

 Otherwise, since it’s not ‘B’ or whitespace

 Skip this character

Print counts

End program

With fgets

#include <stdio.h>

#define ENOUGH\_LINES 1000

char boston[] = “Boston”;

int bcount = 0;

int wcount = 0;

int ws(char c) {

 return c == ‘ ‘ || c == ‘\t’ || c == ‘\n’;

}

void main() {

 int i, j;

 char buffer[ENOUGH\_LINES];

 while (fgets(buffer, ENOUGH\_LINES, stdin)) {

 i = 0;

 while (ws(buffer[i]))

 i++; /\* get rid of leading whitespace \*/

 while(buffer[i] != ‘\0’) {

 if (buffer[i] == boston[0]) { /\* beginning of “Boston”? \*/

 j = 0;

 while (boston[j] && buffer[i] == boston[j]) {

 i++;

 j++;

 if (boston[j] == ‘\0’)

 bcount++; /\* we found Boston \*/

 }

 } else if (ws(buffer[i])) { /\* beginning of whitespace \*/

 while (ws(buffer[i]))

 i++; /\* find next word \*/

 wcount++; /\* count the word we were just on \*/

 } else {

 i++; /\* nothing interesting, next index \*/

 }

 }

 }

 printf(“%d, %d\n”, bcount, wcount);

}

With getchar()

#include <stdio.h>

#define ENOUGH\_LINES 1000

char boston[] = “Boston”;

int bcount = 0;

int wcount = 0;

int ws(char c) {

 return c == ‘ ‘ || c == ‘\t’ || c == ‘\n’;

}

void main() {

 int c = getchar();

 int i;

 while (ws(c))

 c = getchar(); /\* get rid of leading whitespace \*/

 while (c != ‘\0’)

 if (boston[0] == c) {

 i = 0;

 while (boston[i] && boston[i] == c) { /\* beginning of “Boston”? \*/

 i++;

 if (boston[i] == ‘\0’)

 bcount++; /\* we found Boston \*/

 c = getchar();

 }

 } else if (ws(c)) { /\* beginning of whitespace \*/

 while (ws(c))

 c = getchar(); /\* find next word \*/

 wcount++; /\* count the word we were just on \*/

 } else {

 c = getchar(); /\* nothing interesting, next char \*/

 }

 }

}

Note that fgets wasn’t any more helpful here than getchar(). Be ready to decide exactly what functions you need on the exam and pick the right ones for the problem.