Question Unix–1: (Solution, p 5) How would the behavior of the following two Unix shell commands differ?

```
unix% grep open f > wc
unix% grep open f | wc
```

Question Unix–2: (Solution, p 5) The `-w` option for `wc` tells it to count the number of words the command reads from its input. What Unix command would count the number of words occurring in the first two lines of a file named “`data`”?

Question Unix–3: (Solution, p 5) Write a Unix shell command that displays all lines containing the word *Plato* from the first twenty lines of a file called “`phil`.”

Question Unix–4: (Solution, p 5) Write a Unix shell command that displays the last line that contains the word *Plato* from a file called “`phil`”.

Question Unix–5: (Solution, p 5) The “`-1`” option to the `ls` command tells it to list one file name per line; the “`-l`” option for `wc` tells it to count the number of lines it finds. Write a Unix shell command to determine how many files in the current directory contain `210` in their name.

Question Unix–6: (Solution, p 5) The `uniq` Unix command echoes the lines from its input onto its output, but omitting repeated adjacent lines.

```
% cat file
aab
aa
aa
aa
aab
% uniq < file
aab
aa
aab
```

In this example transcript, `uniq` listed the `aa` line once once, even though it occurs three successive times in `file`. It displays `aab` both times because the occurrences are not adjacent.

Suppose “`words`” is an unordered file of words, one per line. What Unix command would count the number of distinct words in this file?

Question 2–1: (Solution, p 5) Translate each of the following Java programs to its closest C equivalent.

a.

```
public class FindZero {
    public static double f(double x) {
        return x * x + 4 * x + 3;
    }

    public static void main(String[] args) {
        double neg = -2; // f(neg) < 0
        double pos =  2; // f(pos) < 0

        while(pos - neg < 0.01
                || pos - neg > 0.01) {
            double mid = (neg + pos) / 2;
            if(f(mid) < 0) {
                neg = mid;
            } else {
                pos = mid;
            }
        }
        System.out.println("f(" + neg
            + ") = " + f(neg));
    }
}
```

b.

```
public class CountEvens {
    public static boolean isEven(int n) {
        if(n % 2 == 0) {
            return true;
        } else {
            return false;
        }
    }

    public static void main(String[] args) {
        int count = 0;
        for(int i = 0; i < 100; i++) {
            if(isEven(i)) count++;
        }
        System.out.println(count);
    }
}
```

Question 2–2: (Solution, p 5) Translate the following Java method into a C function.

```
public static boolean isPalindrome(int[] arr) {
    for(int i = 0; i < 5; i++) {
        if(arr[i] != arr[9 - i]) return false;
    }
    return true;
}
```

Question 2–3: (Solution, p 5) What does each of the following C programs print when run?

a.

```
#include <stdio.h>

int main() {
    int i;  int j;
    int *p; int *q;

    p = &i;
    q = &j;
    i =  9;
    j =  8;
    p =  q;
    *q = 7;
    printf("%d %d\n",
        i, j);
    printf("%d %d\n",
        *p, *q);
    return 0;
}
```

b.

```
#include <stdio.h>

int main() {
    int i;  int j;
    int *p; int *q;

    p = &i;
    q = &j;
    i =  3;
    j =  5;
    *q = 8;
    q =  p;
    printf("%d %d\n",
        i, j);
    printf("%d %d\n",
        *p, *q);
    return 0;
}
```

c.

```
#include <stdio.h>

void mystery(int *ap, int *bp) {
    *ap = *ap + *bp;
    *bp = *ap - *bp;
    *ap = *ap - *bp;
}

int main() {
    int i;
    int j;

    i = 4;
    j = 5;
    mystery(&i, &j);
    printf("%d %d\n", i, j);
    i = 6;
    mystery(&i, &i);   /* Note: both args */
    printf("%d\n", i); /* are now &i      */
    return 0;
}
```

Question 2–4: (Solution, p 6) Write the following C function.

`int strchr(char *search, char find)`

Returns the index of the first occurrence of `find` within `search`. If it does not occur, the function returns −1.

Question 2–5: (Solution, p 6) What does each of the following C programs print when run?

a.

```
#include <stdio.h>

void mystery(char *s) {
    int count = 0;
    for(; *s != ’\0’; s++) {
        if(*s == ’r’) count++;
    }
    return count;
}

int main() {
    printf("%d\n", mystery("redder rudder"));
    return 0;
}
```

b.

```
#include <stdio.h>

void mystery(char *dst, char *src) {
    char *p;

    for(p = src; *p != ’\0’; p++) {
        *dst = ’0’; dst++;
    }
    *dst = ’\0’; dst--;
    for(p = src; *p != ’\0’; p++) {
        if(*p != ’0’) {
            *dst = *p; dst--;
        }
    }
}

int main() {
    char out[100];
    mystery(out, "1002");
    printf("%s\n", out);
    return 0;
}
```

Question 3.1–1: (Solution, p 6) How many bits do you need to represent seven different values? Nine? Twelve? Thirty?

Question 3.1–2: (Solution, p 6) How many bits are in a kilobyte of memory?

Question 3.1–3: (Solution, p 6) Perform each of the following conversions.

a. $101101_{(2)}$ to decimal
b. $1010101_{(2)}$ to decimal
c. $23_{(10)}$ to binary
d. $95_{(10)}$ to binary

Question 3.1–4: (Solution, p 6) Perform each of the following conversions.

a. $1010101_{(2)}$ to octal
b. $1010101_{(2)}$ to hexadecimal
c. $101101_{(2)}$ to hexadecimal
d. $560_{(8)}$ to binary
e. $\mathrm{CAB}_{(16)}$ to binary
f. $\mathrm{1B2}_{(16)}$ to binary

Question 3.2–1: (Solution, p 6) Represent each of the following in a sign-magnitude representation.

a. $-1_{(10)}$ in a seven-bit sign-magnitude format
b. $-20_{(10)}$ in a seven-bit sign-magnitude format
c. $20_{(10)}$ in a seven-bit sign-magnitude format
d. $-300_{(10)}$ in twelve-bit sign-magnitude format

Question 3.2–2: (Solution, p 6) Represent each of the following in a two's-complement representation.

a. $-1_{(10)}$ in a seven-bit two's-complement format
b. $-20_{(10)}$ in a seven-bit two's-complement format
c. $20_{(10)}$ in a seven-bit two's-complement format
d. $-300_{(10)}$ in twelve-bit two's-complement format

Question 3.2–3: (Solution, p 6)

a. What is the smallest (most negative) number you can represent in seven bits using sign-magnitude representation? Give both the bit pattern of the number and its base-10 translation.

b. Answer the same question for a seven-bit two's-complement representation.

Solution Unix–1: (Question, p 1) The “`grep open f > wc`” command would redirect what *grep* would normally print on the screen into a file called *wc*, whereas “`grep open f | wc`” would instead pipe this same information to be input to the command *wc*.

Solution Unix–2: (Question, p 1) **`head -2 data | wc -w`**

Solution Unix–3: (Question, p 1) **`head -20 phil | grep Plato`**

Solution Unix–4: (Question, p 1) **`grep Plato phil | tail -1`**

Solution Unix–5: (Question, p 1) **`ls -1 | grep 210 | wc -l`** or **`ls -1 *210* | wc -l`**

Solution Unix–6: (Question, p 1) **`sort < words | uniq | wc -l`**

Solution 2–1: (Question, p 2)

a.

```
#include <stdio.h>

double f(double x) {
    return x * x + 4 * x + 3;
}

int main() {
    double neg; /* f(neg) < 0 */
    double pos; /* f(pos) < 0 */
    double mid;

    neg = -2;
    pos = 2;
    while(pos - neg < 0.01
            || pos - neg > 0.01) {
        mid = (neg + pos) / 2;
        if(f(mid) < 0) {
            neg = mid;
        } else {
            pos = mid;
        }
    }
    printf("f(%f) = %f\n", neg, f(neg));
    return 0;
}
```

b.

```
#include <stdio.h>

int isEven(int n) {
    if(n % 2 == 0) {
        return 1;
    } else {
        return 0;
    }
}

int main() {
    int count;
    int i;

    count = 0;
    for(i = 0; i < 100; i++) {
        if(isEven(i)) count++;
    }
    printf("%d\n", count);
    return 0;
}
```

Solution 2–2: (Question, p 2)

```
int isPalindrome(int *arr) {
    int i;
    for(i = 0; i < 5; i++) {
        if(arr[i] != arr[9 - i]) return 0;
    }
    return 1;
}
```

Solution 2–3: (Question, p 2)

a.

```
9 7
7 7
```

b.

```
3 8
3 3
```

c.

```
5 4
0
```

Solution 2–4: (Question, p 3)

```
int strchr(char *search, char find) {
    int i;

    for(i = 0; search[i] != ’\0’; i++) {
        if(search[i] == find) return i;
    }
    return -1;
}
```

Solution 2–5: (Question, p 3)

a. `4`

b. `0021`

Solution 3.1–1: (Question, p 3) You need 3 bits for seven values, 4 for nine or twelve, and 5 for thirty values.

Solution 3.1–2: (Question, p 3) There are 8,192 bits in a kilobyte:

$$\frac{8\text{ bits}}{\text{byte}} \times \frac{1{,}024\text{ bytes}}{\text{KB}} = \frac{8{,}192\text{ bits}}{\text{KB}}$$

Solution 3.1–3: (Question, p 3)

a.	$101101_{(2)}$	$=$	$45_{(10)}$
b.	$1010101_{(2)}$	$=$	$85_{(10)}$
c.	$23_{(10)}$	$=$	$10111_{(2)}$
d.	$95_{(10)}$	$=$	$1011111_{(2)}$

Solution 3.1–4: (Question, p 3)

a.	$1\,010\,101_{(2)}$	$=$	$125_{(8)}$
b.	$101\,0101_{(2)}$	$=$	$55_{(16)}$
c.	$10\,1101_{(2)}$	$=$	$2D_{(16)}$
d.	$560_{(8)}$	$=$	$101\,110\,000_{(2)}$
e.	$CAB_{(16)}$	$=$	$1100\,1010\,1011_{(2)}$
f.	$1B2_{(16)}$	$=$	$1\,1011\,0010_{(2)}$

Solution 3.2–1: (Question, p 3)

a.	$-1_{(10)}$	100 0001
b.	$-20_{(10)}$	101 0100
c.	$20_{(10)}$	001 0100
d.	$-300_{(10)}$	1001 0010 1100

Solution 3.2–2: (Question, p 4)

a.	$-1_{(10)}$	111 1111
b.	$-20_{(10)}$	110 1100
c.	$20_{(10)}$	001 0100
d.	$-300_{(10)}$	1110 1101 0100

Solution 3.2–3: (Question, p 4)

a.	**Sign-magnitude:**	111 1111 represents $-63_{(10)}$
b.	**Two’s-complement:**	100 0000 represents $-64_{(10)}$