The exam was a 50-minute, closed-book, closed-notes exam.

**Question 1 Part A**

- $5_b = 111011_b$
- $-1_b = 111111_b$
- $0_b = 000000_b$
- $+0_b = 000000_b$
- $+5_b = 000101_b$

**Question 1 Part B**

Smallest 6-bit number: $100000_b$
Largest 6-bit number: $011111_b$
Negation of smallest 6-bit number: $100000_b$ (or not representable)
Negation of largest 6-bit number: $100001_b$

**Question 1 Part C**

$2^{10}$ is approximately $10^3$

Explanation:

$2^{10} = 1024 \approx 1000 = 10^3$

$2^{64}$ is approximately $1.6 \times 10^{19}$

Explanation:

$2^{64} = 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \times 2^4$
$\approx 10^3 \times 10^3 \times 10^3 \times 10^3 \times 10^3 \times 10^3 \times 16$
$= 16 \times 10^{18}$
$= 1.6 \times 10^{19}$

1.6 * $10^7$ terabytes can be addressed by a pointer that is a 64-bit unsigned number

"Mega" means "million" in standard scientific notation. "Megabyte" might mean $10^6$ bytes or $2^{10}$ bytes. Either one would be close enough to answer this question.

Explanation:

One terabyte = $2^{60}$ bytes
$2^{64} = x \times 2^{60}$
$x = 2^{24}$

So what is $2^{24}$ in standard scientific notation?
$2^{24} = 2^{10} \times 2^{10} \times 2^4$
\[ \approx 10^3 \times 10^3 \times 16 \\
= 16 \times 10^6 \\
= 1.6 \times 10^7 \]

Alternative explanation:

One terabyte \( \approx \) one million megabytes = \( 10^6 \times 10^6 \) bytes = \( 10^{12} \) bytes
\[ 2^{64} \approx x \times 10^{12} \]
\[ 1.6 \times 10^{19} \approx x \times 10^{12} \]
\[ x \approx 1.6 \times 10^7 \]

**Question 2 Part A**

Insert after line 3:

```c
#include <string.h>
```

Delete line 20 and replace with:

```c
char *pcRet = fgets(buffer, 100, stdin);
if ((pcRet == NULL) || (buffer[strlen(buffer)-1] != '\n'))
    exit(EXIT_FAILURE);  /* or return EXIT_FAILURE; */
```

Other correct approaches:

(1) Use a loop of your own instead of calling fgets():

```c
int c;
int i;
for (i = 0; i < 100; i++) {
    c = getchar();
    if (c == EOF) break;
    buffer[i] = (char)c;
}
if (i < 100)
    buffer[i] = '\0';
else
    exit(EXIT_FAILURE);  /* or return EXIT_FAILURE; */
```

(2) Decide that "excessive" means >98, and simply write:

```c
char *pcRet = fgets(buffer, 100, stdin);
if ((pcRet == NULL) || (strlen(buffer) > 98))
    exit(EXIT_FAILURE);  /* or return EXIT_FAILURE; */
```

**Question 2 Part B**

Insert after line 7:

```c
if (*p == '\0') exit(EXIT_FAILURE);
```

**Question 2 Part C**

Delete line 6 and replace with:

```c
unsigned int n = 0;
```
Delete line 12 and replace with:

```c
unsigned int oldn = n;
n *= 10;
if (n/10 != oldn) exit(EXIT_FAILURE);
n += (*p - '0');
if (n < oldn) exit(EXIT_FAILURE);
```

**Question 2 Part D**

/* Return the unsigned int represented by the first sequence of decimal digits in
string p. If there are no decimal digits in string p, then exit with status
EXIT_FAILURE. If the sequence of decimal digits in string p represents an
integer that exceeds the capacity of type unsigned int, then exit with status
EXIT_FAILURE. */

**Question 3 Part A**

parse.h

```c
#ifndef PARSE_INCLUDED
#define PARSE_INCLUDED

unsigned int parse(char *p);
#endif
```

parse.c

```c
#include <ctype.h>

unsigned int parse(char *p) {
    int n = 0;
    while (!isdigit(*p)) {
        p++;
    }
    while (isdigit(*p)) {
        n = n + (*p - '0');
        p++;
    }
    return n;
}
```

strconv.c

```c
#include <parse.h>
#include <stdio.h>

int main(void) {
    char buffer[10];
    unsigned int i;
    gets(buffer);
    i = parse(buffer);
    printf("answer: %u\n", i);
}
return 0;
}

**Question 3 Part B**

```bash
strtoint: strtoint.o parse.o
   gcc217 strtoint.o parse.o -o strtoint

strtoint.o: strtoint.c parse.h
   gcc217 -c strtoint.c

parse.o: parse.c parse.h
   gcc217 -c parse.c
```

Copyright © 2016 by Andrew Appel and Robert M. Dondero, Jr.