Princeton University
COS 217: Introduction to Programming Systems
Spring 2007 Final Exam Answers

The exam was a three-hour, open-book, open-notes exam.

Question 1a

File client.c:

```
#include <stdio.h>
#include <assert.h>
#include <ctype.h>
#include "buffer.h"

#define CONTROL(x) ((x) & 31)

void capitalize_forward(Buffer_T buf)
{
   int size;
   int pos;
   char c;

   size = Buffer_size(buf);
   if (size == 0)
      return;

   pos = Buffer_pos(buf);
   if (pos == size)
      return;

   c = Buffer_getchar(buf, pos);

   if (islower(c)) {
      Buffer_delete(buf);
      c = toupper(c);
      Buffer_insert(buf, c);
   }
   else
      Buffer_forward(buf);
}

void display(Buffer_T buf) {
  int min = Buffer_pos(buf) < 20 ? 0 : Buffer_pos(buf)-20;
  int max = Buffer_size(buf)-Buffer_pos(buf) < 20
            ? Buffer_size(buf) : Buffer_pos(buf)+20;
  int i;
  for (i=min; i<max; i++) {
    if (i==Buffer_pos(buf))
      putchar('|');
    putchar(Buffer_getchar(buf,i));
  }
  if (i==Buffer_pos(buf))
    putchar('|');
  putchar('\n');
}

void run(void) {
  Buffer_T buf = Buffer_new();
  for(;;) {
    char c = getchar();
    switch(c) {
    case CONTROL('f'):
      Buffer_forward(buf);
      break;
```

```
    case CONTROL('b'):
      Buffer_back(buf);
      break;
    case CONTROL('d'):
      Buffer_delete(buf);
      break;
    case CONTROL('h'):
      Buffer_back(buf);
      Buffer_delete(buf);
      break;
    case CONTROL('c'):
      return;
    case CONTROL('k'):     /* new case here */
      capitalize_forward(buf);
      break;
    default:
      if (isprint(c))
         Buffer_insert(buf,c);
    }
    display(buf);
  }
}

#include <termios.h>
#include <unistd.h>

int main(int argc, char **argv) {

  /* Set the input to no-echo, character-at-time ("cbreak")
   * mode, and remember the old mode in t0 */
  struct termios t0, t1;
  tcgetattr(0,&t0);
  t1 = t0;
  t1.c_lflag &= !(ECHO|ICANON);
  tcsetattr(0,0,&t1);

  run();

  /* Set the terminal back to its original mode */
  tcsetattr(0,0,&t0);
  return 0;
}
```

Question 1b

When the front array is empty and the back array is nonempty, the correct behavior is to (1) capitalize the character at the cursor if and only if it is an uppercase letter, and (2) move the cursor forward.

> main() calls run(). run() calls capitalize_forward(). capitalize_forward() calls Buffer_getchar(). Buffer_getchar() returns the first character in the back array. If the character is an uppercase letter, then capitalize_forward() then calls Buffer_delete() to remove that character from the back array, capitalizes the character, and calls Buffer_insert() to insert the character. Doing so moves the cursor past the inserted character. If the character is not an uppercase letter, then capitalize_forward() simply calls Buffer_forward() to move the cursor forward.

When the front array is nonempty and the back array is empty, the correct behavior is to do nothing.

> main() calls run(). run() calls capitalize_forward(). capitalize_forward() calls Buffer_size() and Buffer_pos(), discovers that the values returned by those function calls are equal, and returns.

When the front and back arrays both are empty, the correct behavior is to do nothing.

> main() calls run(). run() calls capitalize_forward(). capitalize_forward() calls Buffer_size(), discovers that the value returned is 0, and returns.

Question 2a

File intmap.h

```
#ifndef INTMAP
#define INTMAP

typedef struct IntMap *IntMap_T;

IntMap_T IntMap_new(void);
void IntMap_free(IntMap_T m);
int IntMap_lookup(IntMap_T m, int i);
void IntMap_update(IntMap_T m, int i, int x);

#endif
```

File intmap.c

```
#include "intmap.h"
#include <stdlib.h>
#include <assert.h>

enum {INITIAL_SIZE = 100};

struct IntMap
{
   int *array;
   int size;
};

IntMap_T IntMap_new(void)
{
   int i;
   IntMap_T m;
   m = (IntMap_T)malloc(sizeof(struct IntMap));
   assert(m != NULL);
   m->array = (int*)malloc(INITIAL_SIZE * sizeof(int));
   assert(m->array != NULL);
   for (i = 0; i < INITIAL_SIZE; i++)
      m->array[i] = 0;
   m->size = INITIAL_SIZE;
   return m;
}

void IntMap_free(IntMap_T m)
{
   if (m == NULL)
      return;
   free(m->array);
   free(m);
}

static void IntMap_expand(IntMap_T m, int newsize)
{
   int i;
   int *newarray;
   assert(m != NULL);
   assert(i >= 0);
   assert(m->size < newsize);
   newarray = (int*)malloc(newsize * sizeof(int));
   for (i = 0; i < m->size; i++)
      newarray[i] = m->array[i];
   for ( ; i < newsize; i++)
      newarray[i] = 0;
   free(m->array);
   m->array = newarray;
}
```

```
int IntMap_lookup(IntMap_T m, int i)
{
   assert(m != NULL);
   assert(i >= 0);
   if (! (i < m->size))
      IntMap_expand(m, i+1);
   return m->array[i];
}

void IntMap_update(IntMap_T m, int i, int x)
{
   assert(m != NULL);
   assert(i >= 0);
   if (! (i < m->size))
      IntMap_expand(m, i+1);
   m->array[i] = x;
}
```

File main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "intmap.h"

int main (int argc, char ** argv) {
   IntMap_T m = IntMap_new();
   int i;
   IntMap_update(m,4,5);
   IntMap_update(m,2,1);
   IntMap_update(m,9,3);
   for (i = 0; i < 12; i++)
      printf("%d ",IntMap_lookup(m,i));
   printf("\n");
   IntMap_free(m);
   return 0;
}
```

Question 2b

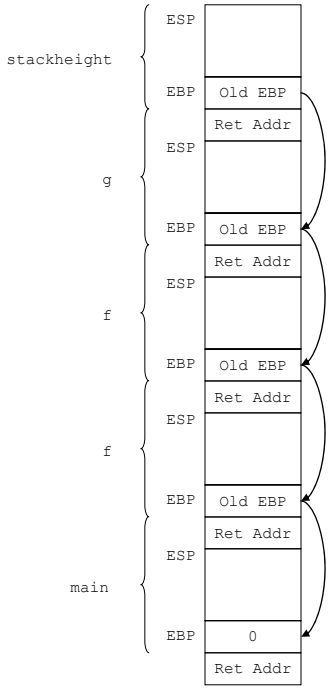
```
CFLAGS = -Wall -ansi -pedantic
# CFLAGS = -Wall -ansi -pedantic -O3

main: main.o intmap.o
    gcc $(CFLAGS) main.o intmap.o -o main

main.o: main.c intmap.h
    gcc $(CFLAGS) -c main.c

intmap.o: intmap.c intmap.h
    gcc $(CFLAGS) -c intmap.c
```

Question 3

There was a small correction to the question. The statement "int i;" should appear at the beginning of the definition of f().

```
        .section ".text"

        .equ N 8
        .equ I -4
        .equ A -44

        .globl f
        .type f, @function

f:
        pushl %ebp
        movl %esp, %ebp

        # int i;
        subl $4, %esp

        # int a[10];
        subl $40, %esp

        # i = 0;
        movl $0, I(%ebp)

loop:
        # if (i > n) goto loopexit;
        movl N(%ebp), %eax
        cmpl %eax, I(%ebp)
        jg loopexit

        # a[i] = n-i;
        movl N(%ebp), %eax
        movl I(%ebp), %ecx
        subl %ecx, %eax
        leal A(%ebp), %edx
        movl %eax, (%edx, %ecx, 4)

        # i++
        incl I(%ebp)

        # goto loop
        jmp loop

loopexit:
        # return g(a) + 3;
        leal A(%ebp), %eax
        pushl %eax
        call g
        addl $4, %esp
        addl $3, %eax

        movl %ebp, %esp
        popl %ebp
        ret
```

Question 4a



Question 4b

```
        .section ".text"

        .globl stackheight
        .type stackheight, @function

stackheight:
        pushl %ebp
        movl %esp, %ebp

        # Set the count to 0.
        movl $0, %eax

        # Fetch the first "old EBP".
        movl (%ebp), %ecx

loop:
        # If the "old EBP" is 0, then jump to loopend.
        cmpl $0, %ecx
        je loopend

        # Increment the count.
        incl %eax

        # Fetch the next "old EBP".
        movl (%ecx), %ecx

        jmp loop

loopend:
        # Return the count (which happens to be in EAX already).
        movl %ebp, %esp
        popl %ebp
        ret
```

Question 5

We are given:

$P(f_{n+1}|ham) = P(f_{n+1}|spam)$

We must prove:

(1) $P(spam|f_1,...\sim f_n,f_{n+1}) = P(spam|f_1,...\sim f_n)$

(2) $P(spam|f_1,...\sim f_n,\sim f_{n+1}) = P(spam|f_1,...\sim f_n)$

Part (1)

$$P(spam|f_1,...\sim f_n,f_{n+1})$$

Use the naïve Bayesian learning algorithm:

$$\approx \frac{P(spam)P(f_1|spam)...P(\sim f_n|spam)P(f_{n+1}|spam)}{P(ham)P(f_1|ham)...P(\sim f_n|ham)P(f_{n+1}|ham) + P(spam)P(f_1|spam)...P(\sim f_n|spam)P(f_{n+1}|spam)}$$

Replace $P(f_{n+1}|ham)$ with $P(f_{n+1}|spam)$:

$$= \frac{P(spam)P(f_1|spam)...P(\sim f_n|spam)P(f_{n+1}|spam)}{P(ham)P(f_1|ham)...P(\sim f_n|ham)P(f_{n+1}|spam) + P(spam)P(f_1|spam)...P(\sim f_n|spam)P(f_{n+1}|spam)}$$

Divide both numerator and denominator by $P(f_{n+1}|spam)$:

$$= \frac{P(spam)P(f_1|spam)...P(\sim f_n|spam)}{P(ham)P(f_1|ham)...P(\sim f_n|ham) + P(spam)P(f_1|spam)...P(\sim f_n|spam)}$$

Use the naïve Bayesian learning algorithm:

$$\approx P(spam|f_1,...\sim f_n)$$

Part (2)

$$P(spam|f_1,...\sim f_n,\sim f_{n+1})$$

Use the naïve Bayesian learning algorithm:

$$\approx \frac{P(spam)P(f_1|spam)...P(\sim f_n|spam)P(\sim f_{n+1}|spam)}{P(ham)P(f_1|ham)...P(\sim f_n|ham)P(\sim f_{n+1}|ham) + P(spam)P(f_1|spam)...P(\sim f_n|spam)P(\sim f_{n+1}|spam)}$$

$P(\sim f_{n+1}|spam) = 1 - P(f_{n+1}|spam) = 1 - P(f_{n+1}|ham) = P(\sim f_{n+1}|ham)$
So replace $P(\sim f_{n+1}|ham)$ with $P(\sim f_{n+1}|spam)$:

$$= \frac{P(spam)P(f_1|spam)...P(\sim f_n|spam)P(\sim f_{n+1}|spam)}{P(ham)P(f_1|ham)...P(\sim f_n|ham)P(\sim f_{n+1}|spam) + P(spam)P(f_1|spam)...P(\sim f_n|spam)P(\sim f_{n+1}|spam)}$$

Divide both numerator and denominator by $P(\sim f_{n+1}|spam)$:

$$= \frac{P(spam)P(f_1|spam)...P(\sim f_n|spam)}{P(ham)P(f_1|ham)...P(\sim f_n|ham) + P(spam)P(f_1|spam)...P(\sim f_n|spam)}$$

Use the naïve Bayesian learning algorithm:

$$\approx P(spam|f_1,...\sim f_n)$$

Question 6

The spammer could create a script that:

(1) Accesses the free e-mail account web site, capturing the CAPTCHA.

(2) Places the CAPTCHA on a web site that offers free pornography, with the caveat that the user must "type the letters you see in this image."

(3) Forwards the letters typed by the user to the e-mail account web site.

(4) Repeats those steps.

Thus the script allows the spammer to sign up for many free e-mail accounts automatically.