COS 217 Final Exam

Princeton University, Spring 2007

May 22, 2007

This exam is **open book, open notes**. You have **3 hours**.

Your name:

Write and sign the honor pledge:

	Score	Possible
1		12
2		15
3		10
4		11
5		10
6		2
Total		60

Question 1 (12 points). We wish to add new functionality to the "one-line emacs" editor (you can find the source code at the end of this exam). When the user presses control-U, the editor should capitalize the letter immediately after the cursor, and move forward. If it's already a capital letter, or if it's not a letter, then leave it alone and move forward. If the cursor is at the end of the buffer, do nothing.

A. Implement "capitalize-forward" for the one-line emacs, as beautifully and cleanly as you can. If you modify existing functions, you can just show the changes if you don't want to copy lots of code; if you add new functions, indicate what module they're in.

B. In each of the following boundary cases, state briefly what the correct behavior is, and give a *brief* explanation to convince me that your your implementation has the correct behavior.

1. "front" array empty, "back" array nonempty
2. "front" array nonempty, "back" array empty
3. "front" and "back" arrays both empty

Question 2 (15 points). An Intmap is a lookup table mapping (nonnegative) integers to integers. The operations are:

```
/* intmap.h */
typedef struct intmap *Intmap;
Intmap Intmap_new(void);
int Intmap_lookup (Intmap m, int i);
void Intmap_update (Intmap m, int i, int x);
void Intmap_free (Intmap m);
```

If I do "update(m,i,x)" and then later I do "lookup(m,i)" then the result will be x (assuming there are no intervening updates at position i). Any lookup to a never-updated element will return zero. Updates to negative "i" values are illegal.

As an example, here's a function that uses the maps:

```
/* main.c */
#include <stdio.h>
#include "intmap.h"

int main (int argc, char ** argv) {
  Intmap m = Intmap_new();
  int i;
  Intmap_update(m,2,2);
  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;
}
```

and its output is `0 0 1 0 5 0 0 0 0 3 0 0`.

A. Implement the abstract data type of integer maps. You must use the following representation: a dynamically allocated array of integers with at least enough slots to hold the largest "i" value ever updated. In addition to this array, you may need a small amount of bookkeeping information. Then, update(m,i,x) will use slot i (if available), or will resize the array as necessary. Use only malloc/free, not calloc or realloc.

Use assertions to enforce design by contract. Do not bother with comments unless you are doing something that you think I wouldn't understand otherwise.

. . . implement the abstract data type of integer maps . . .

B. Show the makefile for the integers maps program. For full credit, make it easy to adjust the compiler-optimization level.

Question 3 (10 points). Translate the function `f` to assembly language. (Using super-fancy addressing modes will not improve your score, nor hurt it.)

```
extern int g(int *p);

int f(int n) {
 int a[10];
 for(i=0; i<=n; i++)
   a[i]=n-i;
 return g(a)+3;
}
```

Question 4 (11 points). Function `main` calls function `f`, which calls `f`, which calls `g`, which calls the `stackheight` function. The `stackheight` function counts the number of stack-frames on the stack, including `main` but not including `stackheight`, and returns that number. In this case it returns 4.

How does `stackheight` work? It starts from its own ebp, and traces saved ebp's until it finds a saved-ebp equal to 0. You can assume that the saved-ebp in `main`'s stack frame is 0.

A. Draw a picture of stack frames, indicating saved ebp's and return addresses, for the situation described (main → f → f → g → stackheight). Draw arrows indicating where the saved ebp's point to. You don't have to indicate all the other goo in the stack frames, just leave blank spaces there.

B. Write the `stackheight` function in assembly language that counts (and returns) the number of frames on the stack.

Question 5 (10 points). Suppose I have a Spam filter that uses naive Bayes machine learning with N features $(f_1, \ldots, f_N)$. I add one more feature, f_{N+1}, but it turns out that my new feature occurs exactly as often in Spam as it does in Ham.

Demonstrate mathematically that my new feature does no harm, that is, for any test message, the $(N+1)$-feature spam filter will report exactly the same answer as the N-feature spam filter.

Question 6 (2 points). CAPTCHA stands for "**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part." When you sign up for something like a free e-mail account, you may be presented with a CAPTCHA, that is, "type the letters you see in this image,"

The purpose of CAPTCHAs is to prevent a spammer from using an automated script to sign up for thousands of free e-mail accounts, because an automated script won't be able to recognize the letters in the images.

A. Explain how spammers can use pornography to defeat the CAPTCHA and sign up for thousands of free e-mail accounts.

END OF EXAM. (6 questions, 60 total points.)

buffer.h:

```
typedef struct buffer *Buffer_T;

Buffer_T Buffer_new(void);

void Buffer_insert(Buffer_T b, char c);
void Buffer_delete(Buffer_T b);

void Buffer_forward(Buffer_T b);
void Buffer_back(Buffer_T b);

int  Buffer_size(Buffer_T b);
int  Buffer_pos(Buffer_T b);
char Buffer_getchar(Buffer_T b, int index);
```

client.c:

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

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

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;
    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;
}
```

buffer2.c:

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

#define CAPACITY  1000

struct buffer {
  int frontsize, backsize;
  char *front, *back;
  int f, b;
};

Buffer_T Buffer_new(void) {
  Buffer_T buf =
       (Buffer_T) malloc (sizeof *buf);
  assert(buf);
  buf->frontsize = buf->backsize = CAPACITY;
  buf->front = (char *) malloc (buf->frontsize);
  assert(buf->front);
  buf->f = buf->b = 0;
  buf->back = (char *) malloc (buf->backsize);
  assert(buf->back);
  return buf;
}

void Buffer_insert(Buffer_T buf, char c) {
  assert(buf);
  assert (buf->f + 1 < buf->frontsize);
  buf->front[buf->f++] = c;
}


void Buffer_delete(Buffer_T buf) {
  assert(buf);
  assert (buf->b > 0);
  --(buf->b);
}

void Buffer_forward(Buffer_T buf) {
  assert(buf);
  assert(buf->f + 1 < buf->frontsize);
  assert(buf->b > 0);
  buf->front[buf->f++] = buf->back[--(buf->b)];
}

void Buffer_back(Buffer_T buf) {
  assert(buf);
  assert(buf->f > 0);
  assert(buf->b + 1 < buf->backsize);
  buf->back[buf->b++] = buf->front[--(buf->f)];
}
```

```
int  Buffer_size(Buffer_T buf) {
  assert(buf);
  return (buf->f + buf->b);
}

char Buffer_getchar(Buffer_T buf, int index) {
  assert(buf);
  assert(index >= 0 && index < buf->f + buf->b);
  if (index < buf->f)
    return (buf->front[index]);
  else return (buf->back[
          buf->f + buf->b - 1 - index]);
}

int  Buffer_pos(Buffer_T buf) {
  assert(buf);
  return (buf->f);
}
```