CMPUT201W20B2 Midterm 2 Practice

Midterm 2 Practice

Copyright Statement

License

Init ORG-MODE

```
;; I need this for org-mode to work well
(require 'ob-sh)
;(require 'ob-shell)
(org-babel-do-load-languages 'org-babel-load-languages '((sh . t)))
;(org-babel-do-load-languages 'org-babel-load-languages '((shell . t)))
(org-babel-do-load-languages 'org-babel-load-languages '((C . t)))
(org-babel-do-load-languages 'org-babel-load-languages '((python . t)))
(setq org-src-fontify-natively t)
```

```
(setq org-confirm-babel-evaluate nil) ;; danger!
(custom-set-faces
 ;; custom-set-faces was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(org-block ((t (:inherit shadow :foreground "black")))))
```

Org export

```
(org-html-export-to-html)
(org-latex-export-to-pdf)
(org-ascii-export-to-ascii)
```

Org Template

Copy and paste this to demo C

```
#include <stdio.h>

int main(int argc, char**argv) {
    return 0;
}
```

Remember how to compile?

gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o programname programname.c

Midterm Practice Questions

The midterm practice questions will be denoted as

```
/* QUESTION
   1: code example 1
   2: code example 2
   3: code example 3
   4: code example 4

   or

   Q: What code should go here?

   You choose. No answers will be given you must compile the code to
   find out.
*/
```

Questions on Flat Allocation

This program tries to calculate the chance of flush hands appearing in poker.

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

enum card_face {
    ACE = 1,
    FACE2,
    FACE3,
    FACE4,
    FACE5,
    FACE6,
    FACE7,
    FACE8,
    FACE9,
    FACE10,
    JACK,
    QUEEN,
    KING,
};
/* QUESTION: What should go here?
   1: Nothing
   2: typedef enum CardFace card_face;
   3: typedef enum card_face CardFace;
   4: typedef CardFace enum card_face;
   5: typedef card_face CardFace;
*/

#define NFACES 13
#define NFACEOFF 1

enum card_suit {
    CLUBS,
    HEARTS,
    DIAMONDS,
    SPADES
};

typedef enum card_suit CardSuit;

#define NSUIT 4

struct playing_card {
    CardFace face;
    CardSuit suit;
};

typedef struct playing_card PlayingCard;

#define HANDSIZE 5

/* QUESTION: What should go for the definition of isFlush that detects
flushes
   1: Nothing
   2: bool isFlush(PlayingCard hand[HANDSIZE]) {
   3: bool isFlush(const PlayingCard hand[HANDSIZE]) {
   4: bool isFlush(PlayingCard ** hand) {
```

```
   5: bool isFlush(const PlayingCard ** hand) {
   6: bool isFlush(PlayingCard *** hand) {
   7: bool isFlush(CardFace * hand) {
   8: bool isFlush(CardSuit * hand) {
   9: bool isFlush(CardSuit suits[HANDSIZE]) {
*/
// missing function defintion
    CardSuit suit = hand[0].suit;
    for (int i = 1;  i < HANDSIZE; i++ ) {
        if (suit != hand[i].suit) {
            return false;
        }
    }
    return true;
}

// This function generates random cards
PlayingCard randomCard() {
    PlayingCard card = {ACE, CLUBS};
/* QUESTION: What should go here to randomize the faces of a card?
    1: Nothing
    2: card->face = CLUBS;
    3: card->face = card->face;
    4: card->face = rand() % NFACES;
    5: card->face = NFACEOFF + ( rand() % NFACES );
    6: card.face = card.face;
    7: card.face = rand() % NFACES;
    8: card.face = NFACEOFF + ( rand() % NFACES );
*/
    card.suit =  rand() % NSUIT;
    return card;
}
int main() {
    srand(time(NULL));
    const int HANDS = 1000000;
/* QUESTION: What should go here to allocate heap memory for HANDS number of
playing card hands?
    1: Nothing
    2: PlayingCard hands[HANDS*HANDSIZE];
    3: PlayingCard hands[HANDS][HANDSIZE];
    4: PlayingCard * hands[HANDS][HANDSIZE];
    5: PlayingCard * hands = malloc(sizeof(PlayingCard)*HANDS*HANDSIZE);
    6: PlayingCard * hands[HANDS][HANDSIZE] =
malloc(sizeof(PlayingCard)*HANDS*HANDSIZE);
    7: PlayingCard (*hands)[HANDS][HANDSIZE] =
malloc(sizeof(PlayingCard)*HANDS*HANDSIZE);
    8: PlayingCard (*hands)[HANDS][HANDSIZE];
    9: PlayingCard **hands;
   10: PlayingCard **hands = malloc(sizeof(PlayingCard)*HANDS*HANDSIZE);
*/
    for (int i = 0; i < HANDS*HANDSIZE; i++) {
        hands[i] = randomCard();
    }
    int flushes = 0;
    for (int i = 0; i < HANDS; i++) {
        if (isFlush(hands + i*HANDSIZE)) {
            if (flushes < 10) { // reduce printing
```

```
                printf("Flush found at card %d\n", i);
                printf("Suit %d\n", hands[i].suit);
            }
            flushes++;
        }
    }
    printf("We found %d flushes out of %d hands: %f\n", flushes, HANDS,
flushes/(float)(HANDS));
}
```

Example output:

```
Flush found at card 19
Suit 3
Flush found at card 340
Suit 1
Flush found at card 450
Suit 0
Flush found at card 870
Suit 0
Flush found at card 918
Suit 1
Flush found at card 932
Suit 2
Flush found at card 970
Suit 2
Flush found at card 1375
Suit 0
Flush found at card 1438
Suit 3
Flush found at card 1631
Suit 2
We found 3902 flushes out of 1000000 hands: 0.003902
```

Questions on Pointers to Pointers

Please finish the prior section before you do this section. This section has answers for the prior section.

This program is the program modified to use pointers to pointers rather than a flat 2D array in memory.

```
  #include <stdbool.h>
  #include <stdio.h>
  #include <stdlib.h>
  #include <time.h>
  #include <assert.h>

  enum card_face {
      ACE = 1,
      FACE2,
      FACE3,
      FACE4,
      FACE5,
      FACE6,
      FACE7,
      FACE8,
      FACE9,
      FACE10,
      JACK,
      QUEEN,
      KING,
  };

  typedef enum card_face CardFace;

  #define NFACES 13
  #define NFACEOFF 1

  enum card_suit {
      CLUBS,
      HEARTS,
      DIAMONDS,
      SPADES
  };

  typedef enum card_suit CardSuit;

  #define NSUIT 4

  struct playing_card {
      CardFace face;
      CardSuit suit;
  };

  typedef struct playing_card PlayingCard;

  #define HANDSIZE 5

  bool isFlush(PlayingCard hand[HANDSIZE]) {
      CardSuit suit = hand[0].suit;
      for (int i = 1;  i < HANDSIZE; i++ ) {
          if (suit != hand[i].suit) {
              return false;
          }
      }
      return true;
  }

  PlayingCard randomCard() {
      PlayingCard card = {ACE, CLUBS};
      card.face = NFACEOFF + ( rand() % NFACES );
      card.suit =  rand() % NSUIT;
      return card;
  }

  // We want to allocate a hand of playing cards (HANDSIZE PlayingCards)
  // on the heap
```
/* QUESTION: What is the definition of the question allocateHand where it is supposed to allocate and return a

```
   hand of PlayingCards?
   1: PlayingCard * allocateHand() {
   2: PlayingCard[] allocateHand() {
   3: PlayingCard[HANDSIZE] allocateHand() {
   4: PlayingCard** allocateHand() {
   5: PlayingCard allocateHand() {
   6: void allocateHand(Playingcard * hand) {
   7: void allocateHand(Playingcard *** hand) {
   8: void allocateHand(Playingcard hand[HANDSIZE]) {
   9: void allocateHand(Playingcard hand[handsize], size_t handsize) {
*/
/* QUESTION: What should we fill in to allocate a hand worth of playing cards
on the heap?
   1: PlayingCard ** hand = malloc(sizeof(PlayingCard[HANDSIZE]));
   2: PlayingCard * hand = malloc(sizeof(PlayingCard[]));
   3: PlayingCard * hand = malloc(sizeof(PlayingCard(*)[5]));
   4: PlayingCard * hand = malloc(sizeof(PlayingCard(*)[HANDSIZE]));
   5: PlayingCard hand[HANDSIZE] = malloc(sizeof(PlayingCard(*)[HANDSIZE]));
   6: PlayingCard * hand = malloc(sizeof(PlayingCard[HANDSIZE]));
   7: Hand hand = malloc(sizeof(Hand));
*/
      assert(hand!=NULL);
      return hand;
  }

  void randomizeHand( PlayingCard hand[HANDSIZE]) {
      for (int i = 0; i < HANDSIZE; i++) {
          hand[i] = randomCard();
      }
  }

  int main() {
      srand(time(NULL));
      const int HANDS = 1000000;
      // Pointer to arrays of arrays
/* QUESTION: how should I allocate pointers to hands which are pointers to
HANDSIZE PlayingCard?
             Choose the most correct!
      1: Nothing
      2: PlayingCard **hands = malloc(sizeof(PlayingCard) * HANDSIZE *
HANDS);
      3: PlayingCard **hands = malloc(sizeof(PlayingCard(*)[HANDSIZE]) *
HANDS);
      4: PlayingCard hands[HANDS][HANDSIZE] =
malloc(sizeof(PlayingCard(*)[HANDSIZE]) * HANDS);
      5: PlayingCard (*hands)[HANDSIZE] =
malloc(sizeof(PlayingCard(*)[HANDSIZE]) * HANDS);
      6: PlayingCard (*hands)[HANDS] = malloc(sizeof(PlayingCard(*)[HANDS]) *
HANDS);
*/
      for (int i = HANDS-1; i >= 0; i--) {
/* QUESTION: What should we do with hand[i]?
      1: Nothing
      2: hands[i][0] = allocateHand();
      3: hands + i = allocateHand();
      4: hands[i] = allocateHand();
      5: hands[i] = { 0, 0, 0, 0, 0};
      6: hands[i] = { randomizeHand(),
randomizeHand(),randomizeHand(),randomizeHand(), randomizeHand()}
      7: hands[i] = { allocateHand(),
allocateHand(),allocateHand(),allocateHand(), allocateHand()}
*/
          randomizeHand( hands[i] );
      }
      int flushes = 0;
      for (int i = 0; i < HANDS; i++) {
          if (isFlush(hands[i])) {
              if (flushes < 10) { // reduce printing
                  printf("Flush found at card %d\n", i);
                  printf("Suit %d\n", hands[i][0].suit);
              }
              flushes++;
          }
      }
      printf("We found %d flushes out of %d hands: %f\n", flushes, HANDS,
flushes/(float)(HANDS));
/* Question: what should go here?
   1: Nothing
   2: free(hands);
   3: free(hands[0]);
   4: free(hands[i]);
   5: #define free_hands
   6: free(food);
*/
      for (int i = 0; i < HANDS; i++) {
/* Question: what should go here?
   1: Nothing
   2: free(hands);
   3: free(hands[0]);
   4: free(hands[i]);
   5: #define free_hands
   5: free(food);
*/
      }
/* Question: what should go here?
   1: Nothing
   2: free(hands);
   3: free(hands[0]);
   4: #define free_hands
   5: free(food);
*/

  }
```

This program should produce this output

```
Flush found at card 148
Suit 0
Flush found at card 792
Suit 2
Flush found at card 845
Suit 1
```

```
Flush found at card 1055
Suit 1
Flush found at card 1152
Suit 3
Flush found at card 1240
Suit 0
Flush found at card 1259
Suit 3
Flush found at card 1873
Suit 1
Flush found at card 2368
Suit 0
Flush found at card 2509
Suit 0
We found 4003 flushes out of 1000000 hands: 0.004003
```

Bubblesort from Rosetta code

GFDL licensed code.

Please fix this bubblesort code.

```
#include <stdio.h>

/* QUESTION: what should the function declaration of bubble sort be?
   1: nothing
   2:  int * bubble_sort( int a[n], int n) {
   3:  int bubble_sort( int a[n], int n) {
   4:  void bubble_sort( char a[n], int n) {
   5:  void bubble_sort( double a[n], int n) {
   6:  void bubble_sort (int *a, int n) {
   7:  void bubble_sort (int **a, int n) {
   8:  void bubble_sort (int *a) {
   9:  void bubble_sort (int (*)a[n],int n) {
*/
    int i, t, j = n, s = 1;
    while (s) {
        s = 0;
        for (i = 1; i < j; i++) {
            if (a[i] < a[i - 1]) {
                t = a[i];
                a[i] = a[i - 1];
                a[i - 1] = t;
                s = 1;
            }
        }
        j--;
    }
}

int main () {
    int a[] = {4, 65, 2, -31, 0, 99, 2, 83, 782, 1};
/* QUESTION: what should the variable n be defined and set to?
    1: double n = 10.0
    2: int n = sizeof(a) / sizeof(a[0]);
    3: int n = sizeof(a);
    4: int n = sizeof(a[0]);
    5: int n = sizeof(a[0])/sizeof(double);
    6: int n = 0;
    7: int n;
*/
    int i;
    // print input
    for (i = 0; i < n; i++) {
        printf("%d%s", a[i], i == n - 1 ? "\n" : " ");
    }
    bubble_sort(a, n);
    // print output
    for (i = 0; i < n; i++) {
        printf("%d%s", a[i], i == n - 1 ? "\n" : " ");
    }
    return 0;
}
```

Example output

```
: 4 65 2 -31 0 99 2 83 782 1
: -31 0 1 2 2 4 65 83 99 782
```

N Queens Problem

How do you N queens on a chessboard of width and height N?

https://en.wikipedia.org/wiki/Eight_queens_puzzle

This code is from Rosetta Code and is under the GFDL 1.2.

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

  int count = 0;
  void solve(int n, int col, int *hist)
  {
          // Print the solution if col == n
          if (col == n) {
                  printf("\nNo. %d\n-----\n", ++count);
                  /* Question: Will the next line print a newline after every
row?
                       1: Yes
                       2: No
                       3: Yes, but this is bad style!
                  */
                  for (int i = 0; i < n; i++, putchar('\n')) {
                          for (int j = 0; j < n; j++) {
                                  putchar(j == hist[i] ? 'Q' : ((i + j) & 1)
? ' ' : '.');
                          }
                  }
                  return;
          }

          for (int i = 0, j = 0; i < n; i++) {
                  // go through each column and see if another queen is there
OR if another queen
                  // could diagonally attack in that position
                  for (j = 0; j < col && !(hist[j] == i || abs(hist[j] - i)
== col - j); j++);
                  // if j is less than col it found a queen that attacks
                  // this a queen at this position then try the next
                  // loop iteration, otherwise we have a solution.
  /* Question: which code implements the above comment
                1: if (j < col) continue;
                2: if (j < col) break;
                3: if (j < col) return;
                4: if (j < col) next;
                5: if (j < col) i++;
  */
                  hist[col] = i;
                  // We're good for this column lets solve for the next
column!
  /* Question: which code implements the above comment
                1: Nothing
                2: continue;
                3: break;
                4: solve(col + 1);
                5: solve(n, col, hist);
                6: solve(n, col + 1, hist);
                7: solve(n, 0, hist);
                8: solve(n, col + 1, &hist);
  */
          }
  }

  int main(int n, char **argv)
  {
          if (n <= 1 || (n = atoi(argv[1])) <= 0) n = 8;
/* Question: how should we allocate an array of n ints on the stack?
       1: Nothing
       2: int * hist = malloc(n);
       3: int * hist = malloc(n*sizeof(n));
       4: int * hist = malloc(n*sizeof(int));
       5: int hist[] = malloc(n*sizeof(int));
       6: int hist[n];
*/
          solve(n, 0, hist);
  }
```

Here's a sample of the output

```
No. 1
-----
Q . . .
 . .Q. .
. . . .Q
 . . Q .
. Q . .
 . . .Q.
.Q. . .
 . Q . .

No. 2
-----
Q . . .
 . . Q .
. . . .Q
 .Q. . .
. . . Q
 . Q . .
.Q. . .
 . .Q. .

... # examples ommitted

No. 92
-----
. . . .Q
 . Q . .
Q . . .
 .Q. . .
. . .Q.
 Q . . .
. . . Q
 . .Q. .
```

Recursive Binary Search

Finish this question about binary search.

Binary search was taught in CMPUT 175 but if you need review here's the algorithm:

https://en.wikipedia.org/wiki/Binary_search_algorithm

From RosettaCode https://rosettacode.org/wiki/Binary_search#C licensed under GFDL 1.2

Answer questions about this recursive binary search implementation:

```
#include <stdio.h>

// record the number of calls to bsearch_r
static int calls = 0;
/* Question: What is the best function definition of bsearch_r?
   bsearch_r has to take in a collection of ints, a query to search for, and
upper and lower bounds.
   1: int bsearch_r (int *a, int query, int lower, int upper) {
   2: int bsearch_r (int a[lower][upper], int query, int lower, int upper) {
   3: int bsearch_r (int *a, int x, int i, int j) {
   4: int bsearch_r (float *a, float query, float lower, float upper) {
   5: void bsearch_r (int *a, int query, int lower, int upper) {
   6: int bsearch_r (int *a, int x, int i, int j) {
```

```
*/
    calls++;
    if (upper < lower) {
        return -1;
    }
    int middle = lower + ((upper - lower) / 2);
    if (a[middle] == query) {
        return middle;
    }
    else if (a[middle] < query) {
/* Question: What is the best way to call bsearch_r when the query is larger
than the
   current middle of the range?
      1: return bsearch_r(query, a, middle, upper + 1);
      2: return bsearch_r(query, a, lower, upper);
      3: return bsearch_r(a, query, lower, upper);
      4: return bsearch_r(a, query, middle + 1, upper);
      5: return bsearch_r(a, query, middle - 1, upper - 1);
      6: return bsearch_r(a, query, middle - 1, upper + 1);
      7: return bsearch_r(a + middle, query, 0, upper - middle);
      8: return bsearch_r(a + middle, query, 0, upper);
      9: return bsearch_r(a + middle, query, lower, upper);
*/
    }
    else {
/* Question: What is the best way to call bsearch_r when the query is smaller
than the
   current middle of the range?
     1: return bsearch_r(a, query, lower, lower - 1);
     2: return bsearch_r(a, x, i, k - 1);
     3: return bsearch_r(a, query, lower, middle - 1);
     4: return bsearch_r(a + lower - middle - 1, query, lower, upper);
     5: return bsearch_r(a + lower, query, 0, middle);
     6: return bsearch_r(a, query, lower, upper - 1);
     7: return bsearch_r(a, query, 0, middle - 1);
     8: return bsearch_r(a, query, lower, middle + 1);
     9: return bsearch_r(a, query, lower, middle);
*/
    }
}

int main () {
    int a[] = {-31, 0, 1, 2, 2, 4, 65, 83, 99, 782};
/* Question: what is the best way to calculate the number of elements in a in
a variable n?
 1: int n = sizeof(a) / sizeof(a[0]);
 2: int n = sizeof(a);
 3: int n = sizeof(a[0]);
 4: int n = sizeof(a / sizeof(a[0]));
 5: int n = sizeof((a /(sizeof a[0]));
 6: int n = sizeof(a) / sizeof(long);
 7: int n = sizeof(a) / sizeof(short);
 8: int n = sizeof(long) / sizeof(a);
*/
    for (int i = 0; i < n; i++) {
        int query = a[i];
        calls = 0;
        int index = bsearch_r(a, query, 0, n - 1);
        printf("%d is at index %d in %d calls\n", query, index, calls);
    }
    return 0;
}
```

This is the example output of the working program.

```
-31 is at index 0 in 3 calls
0 is at index 1 in 2 calls
1 is at index 2 in 3 calls
2 is at index 4 in 1 calls
2 is at index 4 in 1 calls
4 is at index 5 in 3 calls
65 is at index 6 in 4 calls
83 is at index 7 in 2 calls
99 is at index 8 in 3 calls
782 is at index 9 in 4 calls
```