COSC242 - Practical Test III

Overview

This lab is worth 12% of your total mark for COSC242. You have a time limit of 1 hour and 45-minutes to complete the test. You will be randomly assigned three programs that you must complete. The first one will be either A or B, the second will be either C or D, and the third one will be either E or F. You have to complete the programs that you have been assigned. No marks will be awarded for writing a different program. Please bring your ID card to the test, and make sure that you come to your own stream on time.

Preparation

Like the previous practical test you can refer to an electronic copy of the lab book. There are also a number of code listings at the end of this document that you can easily access to help you complete the required tasks. A plain text version of this document will be available during the test so you don't need to be concerned about any unusual characters in the PDF. You have a 1 hour 45 minute time slot within which to complete this test.

Remember that we are not asking you to create your programs for the first time during your allocated slot. The code is all from previous labs, however the output format may be slightly different. We strongly encourage you to rewrite your programs a number of times before sitting the test so that you are prepared for any difficulties that may arise. If you prepare well for this lab then you should find it pretty straightforward.

Note: You are not permitted to access your home directory, nor any other files or computers, nor may you use the internet during this lab.

(A) Hash Tables (4%)

Write a basic hash table implementation like the one used in labs with these differences:

- Only store keys in the hash table.
- Use linear probing rather than double hashing.
- Don't store frequencies (adding a duplicate key should have no effect).
- Don't implement search (just doing insert is enough).
- Print out the entire hash table with your print function using this code:

```
for (i = 0; i < h->capacity; i++) {
   fprintf(stream, "%2d %s\n", i, h->keys[i] == NULL ? "" : h->keys[i]);
}
```

In addition to your htable.h and htable.c files you should also have a mylib.h and mylib.c which contain your getword and emalloc functions.

Your program should be in a directory called `lab22a`, and compile and run with htable-test.c given below.

```
/* htable-test.c */

#include <stdio.h>
#include <stdlib.h>
#include "htable.h"
#include "mylib.h"

int main() {
   int size = 17;
   htable h = htable_new(size);
   char word[80];

   while (getword(word, sizeof word, stdin) != EOF) {
      htable_insert(h, word);
   }
   htable_print(h, stdout);
   htable_free(h);
   return EXIT_SUCCESS;
}
```

(B) Binary Search Trees (4%)

Write a regular binary search tree implementation, not a red-black tree. This should be just like the one used in labs, except that you don't need to implement delete. You still need to write a function which deallocates memory.

In addition to your bst.h and bst.c files you should also have a mylib.h and mylib.c which contain your getword and emalloc functions.

Your program should be in a directory called `lab22b`, and compile and run with bst-test.c given below.

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

void print_key(char *key) {
   printf("%s\n", key);
}

void dosearch(bst b, char *key) {
   if (bst_search(b, key) == 0) {
      printf("%s -- not found\n", key);
   } else {
      printf("%s -- found\n", key);
   }
}

int main(void) {
```

```
   bst b = bst_new();

   printf("inserting d,b,f,a,c,e,g\n");
   b = bst_insert(b, "d");
   b = bst_insert(b, "b");
   b = bst_insert(b, "f");
   b = bst_insert(b, "a");
   b = bst_insert(b, "c");
   b = bst_insert(b, "e");
   b = bst_insert(b, "g");

   printf("inorder traversal\n");
   bst_inorder(b, print_key);

   printf("preorder traversal\n");
   bst_preorder(b, print_key);

   printf("searching\n");
   dosearch(b, "f");
   dosearch(b, "o");
   dosearch(b, "x");
   dosearch(b, "e");
   dosearch(b, "d");

   bst_free(b);
   return EXIT_SUCCESS;
}
```

(C) Circular Array Queue (3%)

Write a queue implementation (queue-array.c) which uses a circular array to hold up to 7 doubles. Use the following code as a basis for your solution.

```
#include <stdio.h>
#include <stdlib.h>
#include "mylib.h"
#include "queue.h"

struct queue {
};

queue queue_new() {
   int default_size = 7;
}

void enqueue(queue q, double item) {
   if (q->num_items < q->capacity) {
      q->items[(q->head + q->num_items++) % q->capacity] = item;
   }
}

double dequeue(queue q) {
}
```

```
void queue_print(queue q) {
   /* print queue contents one per line to 2 decimal places */
}

void queue_print_info(queue q) {
   int i;
   printf("capacity %d, num_items %d, head %d\n[", q->capacity,
         q->num_items, q->head);
   for (i = 0; i < q->capacity; i++) {
      printf("%s%.2f", i == 0 ? "" : ", ", q->items[i]);
   }
   printf("]\n");
}

int queue_size(queue q) {
}

queue queue_free(queue q) {
}
```

Your program should be in a directory called `lab22c`, and compile and run with queue-test.c given at the start of the appendix.

(D) Linked List Queue (3%)

Write a queue implementation (queue-llist.c) which uses a linked list to hold doubles. Use the following code as a basis for your solution.

```
#include <stdio.h>
#include <stdlib.h>
#include "mylib.h"
#include "queue.h"

typedef struct q_item *q_item;

struct q_item {
};

struct queue {
};

queue queue_new() {
}

void enqueue(queue q, double item) {
   q_item i = emalloc(sizeof *i);
   i->item = item;
   i->next = NULL;
   if (q->length == 0) {
      q->first = i;
   } else {
      q->last->next = i;
   }
   q->last = i;
```

```
        q->length++;
    }

    double dequeue(queue q) {
    }

    void queue_print(queue q) {
        /* print queue contents one per line to 2 decimal places */
    }

    void queue_print_info(queue q) {
        if (q->length == 0) {
            printf("The queue is empty\n");
        } else {
            printf("first %.2f, last %.2f, length %d\n", q->first->item,
                   q->last->item, q->length);
        }
    }

    int queue_size(queue q) {
    }

    void queue_free_aux(q_item i) {
    }

    queue queue_free(queue q) {
    }
```

Your program should be in a directory called `lab22d`, and compile and run with queue-test.c given at the start of the appendix.

(E) Breadth First Search (5%)

Write a graph implementation as described in lab 19 and perform a breadth first search as specified on page 95-97 of the lab book. You can use a modified version of your queue from part C or D of this test when implementing your breadth first search. You can use the code in `graph-skel.c` as a basis for your solution.

Your program should be in a directory called `lab22e`, and compile and run with `graph-test.c` given in the appendix. You will need to add code `graph-test.c` to perform a breadth first search starting from node 1 and then print the state of your graph.

(F) Depth First Search (5%)

Write a graph implementation as described in lab 19 and perform a depth first search as outlined in lab 20. You can use the code in `graph-skel.c` as a basis for your solution.

Your program should be in a directory called `lab22f`, and compile and run with `graph-test.c` given in the appendix. You will need to add code `graph-test.c` to perform a depth first search and then print the state of your graph.

(G) Graph Data Structure (2%)

If you are unable to successfully implement a solution to the graph searching algorithm that you have been assigned in part E or F but do have a working graph implementation you can get partial marks by following these steps:

- Change into your home directory and create a link to your graph implementation using either the command

  ```
  ln -s lab22e lab22g
  ```

  or the command

  ```
  ln -s lab22f lab22g
  ```

- You can then change into your `lab22g` directory and use the marking script to check your program as usual.
- Your program needs to print out an adjacency list for the input graph.

Marking

- Make sure that no warnings are issued when compiling with the usual compiler flags.
- There is a script called `check-prac-242` which you can run to check that your program is working correctly for a number of different input files (some tests will only be run when on Linux, not OSX). Note that this only checks your program's output. It doesn't check that all of the requirements are fullfilled.
- Once your programs are finished, pass the test script, and meet all of the specifications, raise your hand and a demonstrator will check your work and mark you off.

Appendix

Your queue implementation should compile and run with queue-test.c.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "queue.h"

int main() {
   queue q = queue_new();
   char c;
   double num;

   while (1 == scanf(" %c", &c)) {
      if (c == 'p') {
         queue_print(q);
      } else if (c == 'i') {
         queue_print_info(q);
      } else if (c == 'r' && queue_size(q) > 0) {
         printf("%.2f\n", dequeue(q));
      } else if (c == 'a' && 1 == scanf("%lg", &num)) {
         enqueue(q, num);
      }
   }
   q = queue_free(q);

   return EXIT_SUCCESS;
}
```

```
#ifndef QUEUE_H_
#define QUEUE_H_

typedef struct queue *queue;

queue  queue_new();
queue  queue_free(queue q);
void   enqueue(queue q, double item);
double dequeue(queue q);
void   queue_print(queue q);
void   queue_print_info(queue q);
int    queue_size(queue q);

#endif
```

Your graph implementation should compile and run with graph-test.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include "graph.h"

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

```
    int size, v1, v2;
    int bidirectional = 0, bfs = 0, dfs = 0;
    graph g;
    const char *optstring = "bd2";
    char option;

    while ((option = getopt(argc, argv, optstring)) != EOF) {
        switch (option) {
            case 'b':
                bfs = 1;
                break;
            case 'd':
                dfs = 1;
                break;
            case '2':
                bidirectional = 1;
                break;
        }
    }
    if (1 == scanf("%d", &size)) {
        g = graph_new(size);
        while (2 == scanf("%d %d", &v1, &v2)) {
            if (bidirectional) {
                graph_add_2edges(g, v1, v2);
            } else {
                graph_add_edge(g, v1, v2);
            }
        }
        graph_print_list(g);
        printf("\n");

        if (bfs) { }

        if (dfs) { }

        g = graph_free(g);
    }
    return EXIT_SUCCESS;
}
```

```
/* graph-skel.c */

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "graph.h"
#include "mylib.h"
#include "queue.h"

typedef enum state {UNVISITED, VISITED_SELF, VISITED_DESCENDENTS} state_t;

graph graph_new(int num_vertices) { }

graph graph_free(graph g) { }

void graph_add_2edges(graph g, int u, int v) { }
```

```
void graph_add_edge(graph g, int u, int v) { }

void graph_print_list(graph g) { }

void graph_print_state(graph g) { }

/* either */
void graph_bfs(graph g, int source) { }

/* or */
void graph_dfs(graph g) { }
```

Here is some code from the lab book that you might find useful.

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

int getword(char *s, int limit, FILE *stream) {
   int c;
   char *w = s;
   assert(limit > 0 && s != NULL && stream != NULL);

   /* skip to the start of the word */
   while (!isalnum(c = getc(stream)) && EOF != c)
      ;
   if (EOF == c) {
      return EOF;
   } else if (--limit > 0) {   /* reduce limit by 1 to allow for the \0 */
      *w++ = tolower(c);
   }
   while (--limit > 0) {
      if (isalnum(c = getc(stream))) {
         *w++ = tolower(c);
      } else if ('\'' == c) {
         limit++;
      } else {
         break;
      }
   }
   *w = '\0';
   return w - s;
}
```

```
#ifndef HTABLE_H_
#define HTABLE_H_

#include <stdio.h>

typedef struct htablerec *htable;

extern void     htable_free(htable h);
extern int      htable_insert(htable h, char *str);
extern htable   htable_new(int capacity);
```

```
extern void     htable_print(htable h, FILE *stream);
extern int      htable_search(htable h, char *str);

#endif
```

```
#include <stdio.h>
#include <string.h>
#include "htable.h"
#include "mylib.h"

struct htablerec {
  int capacity;
  int num_keys;
  char **keys;
};

static unsigned int htable_word_to_int(char *word) {
  unsigned int result = 0;

  while (*word != '\0') {
     result = (*word++ + 31 * result);
  }
  return result;
}

static unsigned int htable_hash(htable h, unsigned int i_key) {
  return i_key % h->capacity;
}

htable htable_new(int capacity) { }

void htable_free(htable h) { }

int htable_insert(htable h, char *key) { }

void htable_print(htable h, FILE *stream) { }
```

```
#ifndef BST_H_
#define BST_H_

typedef struct bstnode *bst;

extern bst    bst_free(bst b);
extern void   bst_inorder(bst b, void f(char *str));
extern bst    bst_insert(bst b, char *str);
extern bst    bst_new();
extern void   bst_preorder(bst b, void f(char *str));
extern int    bst_search(bst b, char *str);

#endif
```