COSC242 - Practical Test II

Overview

This lab is worth 8% of your total mark for COSC242. Each lab stream has been divided up into 45-minute slots, during one of which you will sit the practical test. You will be randomly assigned one of the first three programs (A, B, or C) and one of the last two programs (D, or E) to complete. 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 you are on time.

Preparation

Like lab 7, this lab is summative rather than formative in nature. It once again allows you (and us) to look back and see what knowledge and practical skills you have gained at this point of the course. Unlike 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. You have a 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 will 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) Printing prime numbers (3%)

Write a program (in a `lab13a` directory) that prints out the first 200 prime numbers. The primes should be printed in 10 columns, right-justified, with a field width of 5. So your output should start like this:

```
 2    3    5    7   11   13   17   19   23   29
31   37   41   43   47   53   59   61   67   71
...
```

Here is some pseudocode that may help you.

```
is_prime (candidate) {
  for each number n from 2 up to candidate {
    if (candidate divided by n leaves no remainder) return 0
  }
  return 1
```

```
}

main {
  initialise candidate to 2
  initialise num_printed to 0
  while (num_printed is less than 200) {
    if (is_prime(candidate)) {
      print candidate
      increment num_printed
    }
    increment candidate
  }
}
```

(B) Skating scores (3%)

In a certain figure-skating competition, there are three judges who give out scores from 0.0 to 6.0. The contestant's score is worked out by throwing away the smallest score and averaging the other two. A program which accepts three numbers and prints out the competitor's score might look something like this:

```
main {
  declare variables s1, s2 and s3 to hold the judge's scores
  read the 3 scores into s1, s2 and s3
  if (s1 is the lowest score) {
    print (s2 + s3) / 2
  } else if (s2 is the lowest score) {
    print (s1 + s3) / 2
  } else {
    print (s1 + s2) / 2
  }
}
```

You are going to be given input which looks like this:

```
44  3.1  5.0  4.7
21  4.5  5.1  5.8
...
```

where the first number is each competitor's registration number and the rest are the three judges' scores. Write a program that takes any number of lines of input and prints the number of the winning competitor (i.e. who got the largest score as defined above) to stdout. Ignore ties. You can read 4 inputs into 4 variables with

```
scanf("%d%lg%lg%lg", &n, &a, &b, &c)
```

Data should be read from standard input. Your program should be in directory called `lab13b`.

(C) Display random repeats (3%)

Let's say you wanted a program which read a number from standard input, created an array of that size, and then filled it with random numbers. You could write something like this:

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

int main(void) {
  int array_size = 0;
  int *my_array;
  int i = 0;

  printf("Enter the size of the array: ");
  scanf("%d", &array_size);

  my_array = malloc(array_size * sizeof my_array[0]);
  if (NULL == my_array) {
    fprintf(stderr, "memory allocation failed!\n");
    return EXIT_FAILURE;
  }

  for (i = 0; i < array_size; i++) {
    my_array[i] = rand() % array_size;
  }
  printf("What's in the array:\n");
  for (i = 0; i < array_size; i++) {
    printf("%d ", my_array[i]);
  }
  printf("\n");

  free(my_array);

  return EXIT_SUCCESS;
}
```

Extend the program above so that it prints out another line—I want to know which numbers have been repeated in my "random" array.

Write a function which takes the array (and its size) as an argument, and prints out which values in it are repeated (and how many times).

You know that all your random integers have values between 0 and the array size - 1 inclusive. Create an array whose purpose is to keep track of how many times each number appears. Remember to deallocate memory when you are finished.

Your program should be in a directory called `lab13c` and print "X occurs N times" for each repeated number.

(D) Sorting Numbers With Insertion Sort (5%)

Write a program which reads an unknown number of integers from stdin, sorts them into ascending order, and then prints them out, one per line, to stdout.

Specific requirements

Your program must also meet these requirements.

1. Use a flexarray ADT, as demonstrated in lab 9, to do most of the work.
2. Use an erealloc function to dynamically expand the amount of memory allocated. Don't use a separate mylib module like you did in the lab. Just put any functions like erealloc into your flexarray.
3. Print the contents of your array one-per-line to stderr when it is half sorted.
4. Deallocate all allocated memory before the program finishes.
5. Your program should be in a `lab13d` directory, and consist of three files: `sortnums.c`, `flexarray.c`, `flexarray.h`.
6. Use insertion sort to sort the numbers.

(E) Sorting Numbers With Selection Sort (5%)

Write a program which reads an unknown number of integers from stdin, sorts them into ascending order, and then prints them out, one per line, to stdout.

Specific requirements

Your program must also meet these requirements.

1. Use a flexarray ADT, as demonstrated in lab 9, to do most of the work.
2. Use an erealloc function to dynamically expand the amount of memory allocated. Don't use a separate mylib module like you did in the lab. Just put any functions like erealloc into your flexarray.
3. Print the contents of your array one-per-line to stderr when it is half sorted.
4. Deallocate all allocated memory before the program finishes.
5. Your program should be in a `lab13e` directory, and consist of three files: `sortnums.c`, `flexarray.c`, `flexarray.h`.
6. Use selection sort to sort the numbers.

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 pass when running 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

Here is some code that you might find useful.

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

int sum_of_squares(int x, int y) {
  printf("Calculating the sum of squares of %d and %d\n", x, y);
  return x * x + y * y;
}

double sum_of_sqrts(double d, double e) {
  printf("Calculating the sum of square roots of %f and %f\n", d, e);
  return sqrt(d) + sqrt(e);
}

void print_bigger_arg(int x, double d) {
  printf("This function doesn't return anything;\n");
  printf("It just prints out whichever argument is bigger.\n");
  printf("The biggest argument is ");
  if (x > d) {
    printf("%d\n", x);
  } else {
    printf("%f\n", d);
  }
}

int main(void) {
  int first_result;
  double second_result;

  first_result = sum_of_squares(3, 4);
  printf("result = %d\n", first_result);
  second_result = sum_of_sqrts(3.0, 4.0);
  printf("result = %f\n", second_result);
  print_bigger_arg(3, 4.0);

  return EXIT_SUCCESS;
}
```

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

int main(void) {
  double entry = 0.0;
  double total = 0.0;
  int number_of_entries = 0;

  while (1 == scanf("%lg", &entry)) {
    total += entry;
    number_of_entries++;
  }
  /* print the average of all the entries */
```

```
  printf("%f\n", total / number_of_entries);

  return EXIT_SUCCESS;
}
```

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

#define ARRAY_SIZE 10

void multiply_by_3(int *a, int n) {
  int i;

  for (i = 0; i < n; i++) {
    a[i] *= 3;
  }
}

int main(void) {
  int my_array[ARRAY_SIZE];
  int counter = 0;
  int i;

  while (counter < ARRAY_SIZE && 1 == scanf("%d", &my_array[counter])) {
    counter++;
  }

  multiply_by_3(my_array, counter);
  for (i = 0; i < counter; i++) {
    printf("%d\n", my_array[i]);
  }
  return EXIT_SUCCESS;
}
```

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

/*
 * "swap" wants the memory addresses of two integers
 */
void swap(int *x, int *y) {
  int temp = *x;   /* temp gets the value living at memory address x.       */
  *x = *y;         /* the value at x gets the value at y.                   */
  *y = temp;       /* the value at y gets the variable "temp"               */
}

int main(void) {
  int a = 3, b = 4;

  printf("a = %d, b = %d\n", a, b);
  swap(&a, &b); /* pass the addresses of a and b */
  printf("a = %d, b = %d\n", a, b);
  return EXIT_SUCCESS;
}
```

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

#define ARRAY_MAX 10

void insertion_sort(int *a, int n) {
    ; /* Sorting code goes here */
}

int main(void) {
  int my_array[ARRAY_MAX];
  int i, count = 0;

  while (count < ARRAY_MAX && 1 == scanf("%d", &my_array[count])) {
    count++;
  }
  insertion_sort(my_array, count);

  for (i = 0; i < count; i++) {
    printf("%d\n", my_array[i]);
  }

  return EXIT_SUCCESS;
}
```

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

static void array_print(int *a, int n) {
  int i;

  for (i = 0; i < n; i++) {
    printf("%d\n", a[i]);
  }
}

int main(void) {
  int capacity = 2;
  int itemcount = 0;
  int item;
  int *my_array = malloc(capacity * sizeof my_array[0]);

  if (NULL == my_array) {
    fprintf(stderr, "memory allocation failed.\n");
    exit(EXIT_FAILURE);
  }

  while (1 == scanf("%d", &item)) {
    if (itemcount == capacity) {
      capacity += capacity;
      my_array = realloc(my_array, capacity * sizeof my_array[0]);
      if (NULL == my_array) {
        fprintf(stderr, "memory reallocation failed.\n");
        exit(EXIT_FAILURE);
      }
```

```
    }
    my_array[itemcount++] = item;
  }

  array_print(my_array, itemcount);
  free(my_array);

  return EXIT_SUCCESS;
}
```

```
#ifndef FLEXARRAY_H_
#define FLEXARRAY_H_

typedef struct flexarrayrec *flexarray;

extern void        flexarray_append(flexarray f, int item);
extern void        flexarray_free(flexarray f);
extern flexarray   flexarray_new();
extern void        flexarray_print(flexarray f);
extern void        flexarray_sort(flexarray f);

#endif
```

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

struct flexarrayrec {
  int capacity;
  int itemcount;
  int *items;
};

flexarray flexarray_new() {
   /* initialise flexarray structure (including items array) */
}

void flexarray_append(flexarray f, int num) {
   /* add an item to the flexarray, expanding as necessary */
}

void flexarray_print(flexarray f) {
  /* a "for" loop to print out each cell of f->items */
}

void flexarray_sort(flexarray f) {
  /* sort into ascending order */
}

void flexarray_free(flexarray f) {
  /* free the memory associated with the flexarray */
}
```