The University of Melbourne
School of Computing and Information Systems

COMP10002
Foundations of Algorithms
Sample Exam 1 – Answers, Semester 1, 2018

Student ID:

Total marks: 60 **Length:** this exam has 11 pages

Reading Time: fifteen minutes **Writing Time:** two hours

Identical Examination Papers: none **Common Content Papers:** none

Authorised Materials:

- Writing materials, e.g., pens, pencils, are allowed.
- Books, calculators, and dictionaries are not allowed.

Instructions to Invigilators:

- Supply students with standard script book(s).
- The exam paper must be returned with all the written script book(s) to the subject coordinator.

Instructions to Students:

- Answer all questions. *All answers are to be written in the script book(s).*
- You may answer the questions in any order. However, you should write all of the answers that belong to the same question together.
- You are not required to write comments in any of your code fragments or functions except when you are explicitly asked to. If a question says "write a function", you may write relevant further functions if you believe that a decomposition of the problem is relevant.
- You may make use of library functions except when their use is explicitly prohibited. If you do make use of library functions, you must add suitable `#include` lines at the start of each corresponding answer.
- Constants should be `#define`'d prior to use, when appropriate.

Short answer questions [15 marks in total]

1.1 [2 marks] In a 16-bit two's complement number representation for integers, what bit patterns represent the decimal numbers 123 and -123, respectively?

Answer: 0000000001111011, 1111111110000101.

1.2 [2 marks] State two desired properties of numeric processing algorithms (among those listed in the lecture slides) that are different from the desired properties of symbolic processing algorithms.

Answer: (1) Effective, in that yield correct answers and have broad applicability and/or limited restrictions on use. (2) (For approximations) Stable and reliable in terms of the underlying arithmetic being performed.

1.3 [2 marks] State two facilities provided by the C preprocessor.

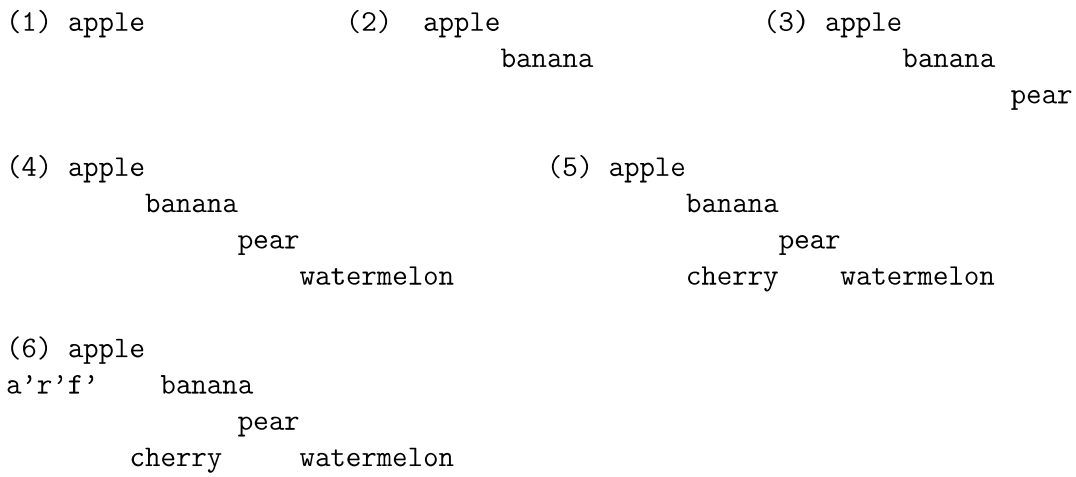
Answer: (1) Symbolic substitution. (2) Conditional compilation, etc.

1.4 [6 marks] You are given an empty binary search tree `T` and an array of strings:

{"apple", "banana", "pear", "watermelon", "cherry", "algorithms are fun"}

Draw the tree `T` after each of the strings is inserted into it.

Answer:



Note:
(1) a'r'f' = algorithms are fun;
(2) also need to draw lines to connect every tree node with its child node(s).

1.5 [3 marks] You are given a pattern string `str` = "apple's app store". Write the failure function values `F[0], F[1], ..., F[16]` corresponding to the pattern string.
Answer:

```
 a p p l e ' s   a p p   s t o r e
-1 0 0 0 0 0 0 0 0 1 2 3 0 0 0 0 0
```

Programming and algorithm questions [45 marks in total]

2. [5 marks]

Write a function

`void reverse_array(int array[], int n)`

that reverses the order of `n` integers in `array`.

For example, if `array = {1, 3, 8, 6, 2}` and `n = 5` is given to the function, then after calling the function, `array` should become `{2, 6, 8, 3, 1}`.

You may assume that `array` contains at least one integer. You may NOT define any new arrays in the `reverse_array()` function (that is, you must operate on `array` itself only).

Answer:

```
void reverse_array(int array[], int n){
    int i, tmp;

    for (i = 0; i < n/2; i++) {
        tmp = array[i];
        array[i] = array[n-1-i];
        array[n-1-i] = tmp;
    }
}
```

3. [5 marks]

Write a function

`int most_frequent(int array[], int n)`

that returns the number that appears for the most times in an array `array` of `n` integers. If there is a tie, return the smaller number.

For example, if `array = {1, 3, 2, 8, 3, 6, 2, 3}` and `n = 8`, then the function should return 3; if `array = {1, 2, 3, 2, 8, 3, 6, 2, 3}` and `n = 9`, then the function should return 2.

You may assume that `array` contains at least one integer.

Answer:

```
#define NO_COUNT_YET -1

int most_frequent(int array[], int n) {
    int i, j, count, max_count = NO_COUNT_YET, max_index;

    for (i = 0; i < n; i++) {
        count = 1;
        for (j = i+1; j < n; j++) {
            if (array[i] == array[j]) {
                count++;
            }
        }
        if (count > max_count) {
            max_count = count;
            max_index = i;
        } else if (count == max_count && array[i] < array[max_index]) {
            max_index = i;
        }
    }

    return array[max_index];
}
```

4. [10 marks in total]

4.1 [7 marks] Write a function

`void my_str_cat(char *dst, char *src)`

that appends string `src` to the end of string `dst`.

For example, if `dst` = "`abc`" and `src` = "`def`", then the call `my_str_cat(dst, src)` will change `dst` to "`abcdef`".

You may assume that `dst` is not `NULL`, and that it has sufficient space to store any new character from `src`. You may NOT change `src` or make use of any functions in the `<string.h>`.

Answer:

```
void my_str_cat(char *dst, char *src) {
    if (src == NULL) {
        return;
    }

    int i = 0, j = 0;

    while(dst[i]) {
        i++;
    }

    while(src[j]) {
        dst[i] = src[j];
        i++;
        j++;
    }
    dst[i] = '\0';
}
```

4.2 [3 marks] Analyse the time complexity of the `my_str_cat()` function.

Answer: Let the length of `dst` and `src` be m and n. The function takes `O(m)` time to get to the end of `dst`, and `O(n)` time to copy `src` to the end of `dst`. Overall, the function takes `O(m+n)` time to run.

5. [15 marks in total]

5.1 [5 marks] Write a function

`int intersect(rectangle_t rect1, rectangle_t rect2)`

that returns `1` if the two rectangles `rect1` and `rect2` interest each other, and `0` otherwise.

Here, each rectangle is represented by the lower and upper bounds in the x-dimension and the lower and upper bounds in the y-dimension, denoted by `lx, ux, ly, uy`, respectively. These bounds should be stored by `int` typed variables.

You need to first write out the definition of the `rectangle_t` type.

To check whether `rect1` and `rect2` intersect, your function `intersect()` needs to check whether the bounds of the two rectangles overlap with each other in both dimensions. Note that if the two rectangles only overlap at a vertex or an edge, they are still considered intersecting.

Answer:

```
typedef struct {
    int lx, ux, ly, uy;
} rectangle_t;

int intersect(rectangle_t rect1, rectangle_t rect2) {
    return !(rect1.ux < rect2.lx || rect1.lx > rect2.ux ||
        rect1.uy < rect2.ly || rect1.ly > rect2.uy);
}
```

5.2 [**5 marks**] Now write another function

`int rect_cmp(void *rect1, void *rect2)`

that compares two rectangles `rect1` and `rect2` by their size.

If `rect1` is smaller than `rect2` in size, then `rect_cmp(rect1, rect2)` should return `1`. If `rect1` is larger than `rect2` in size, then `rect_cmp(rect1, rect2)` should return `-1`. If the two rectangles have the same size, then `rect_cmp(rect1, rect2)` should return `0`.

You may assume that neither pointer is NULL.

Answer:

```
int rect_cmp(void *rect1, void *rect2) {
    rectangle_t *p1 = (rectangle_t *)rect1, *p2 = (rectangle_t *)rect2;

    int area1 = (p1->ux - p1->lx) * (p1->uy - p1->ly),
    int area2 = (p2->ux - p2->lx) * (p2->uy - p2->ly);

    if (area1 < area2) {
        return 1;
    }
    if (area1 > area2) {
        return -1;
    }
    return 0;
}
```

5.3 [**5 marks**] You are given the following type definitions:

```
typedef struct node node_t;
typedef rectangle_t data_t;

struct node {
    data_t data;
    node_t *next;
};
typedef struct {
    node_t *head;
    node_t *foot;
} list_t;

list_t *make_empty_list(void);
void insert_at_head(list_t *list, data_t value);
void insert_at_foot(list_t *list, data_t value);
```

Write a `main()` function that first creates an empty list, then repeatedly prompts a user to input the four bounds of a rectangle and inserts the rectangle to the end of the list until no more input has been entered by the user. The `main()` function should then print out the rectangles in the list from the `head` to the `foot`, one at a line, and then frees the memory allocated for the list.

Answer:

```
int
main(int argc, char *argv[]) {
    list_t *list = make_empty_list();
    rectangle_t rect;
    node_t *head, *old_head;

    while(scanf(``%d %d %d %d'', &rect.lx, &rect.ux, &rect.ly, &rect.uy)
        == 4) {
        insert_at_foot(list, rect);
    }

    head = list->head;
    while(head) {
        printf(``%d %d %d %d'', head->data.lx, head->data.ux,
            head->data.ly, head->data.uy);
        old_head = head;
        head = head->next;
        free(old_head);
    }

    free(list);
    return 0;
}
```

6. [5 marks]

Write a recursive function

`int is_palindrome(char *str, int n)`

that returns `1` if `str` is a palindrome, that is, reads exactly the same forwards as well as backwards. If `str` is not a palindrome, then the function should return `0`. Here, `n` is the length of the string `str`.

For example, if `str =` "`rats live on no evil star`", then `is_palindrome(str, 25)` should return `1`. If `str =` "`abab`", then `is_palindrome(str, 4)` should return `0`.

If you use iteration rather than recursion to answer this question, the full mark of this question will reduce to 2 marks.

Answer:

```
int is_palindrome(char *str, int n) {
    if (str == NULL) {
        return 0;
    }
    if (n <= 1) {
        return 1;
    }
    if (str[0] == str[n-1]) {
        return isPalindrome(str+1, n-2);
    }
    return 0;
}

/* Iterative solution */
int is_palindrome(char* str, int n) {
    if (str == NULL) {
        return 0;
    }
    for (i = 0; i < n/2; i++) {
        if (str[i] != str[n-1-i]){
            return 0;
        }
    }
    return 1;
}
```

7. [5 marks]

Write a function

`int randomised_subset_sum(int items[], int n, int k)`

that uses a randomised strategy to solve the subset sum problem with the set of `n` items represented by the items array, and the sum to achieve represented by `k`.

This randomised strategy repeats the following steps for 10,000 iterations. At each iteration, it randomly chooses an integer `num (0 < num <= n)`. Then it randomly chooses `num` items from the items array. If these `num` items add to `k`, then `randomised_subset_sum()` returns `1`. Otherwise, it starts the next iteration. When 10,000 iterations are completed, `randomised_subset_sum()` returns `0`.

When choosing the `num` items randomly, you need to find a way to guarantee that no item is chosen twice from the items array.

You need to add suitable `#include` lines if you use any library functions.

Answer:

```
#include <stdlib.h>
#include <assert.h>
#define MAX_ITERATION 10000
#define SEED 123456789

int randomised_subset_sum(int items[], int n, int k) {
    int i = 0, num, sum;
    srand(SEED);
    while (i < MAX_ITERATION) {
        num = rand() % n + 1;
        sum = sum_items(items, n, num);
        if (sum == k) {
            return 1;
        }
        i++;
    }
    return 0;
}
```

```
int sum_items(int items[], int n, int num) {
    /* An array to record if an item has been chosen */
    int *flag = (int *)malloc(sizeof(int)*n);
    assert(flag);
    int i, j, counter, sum = 0, choice;

    /* No item has been chosen yet */
    for (i = 0; i < n; i++) {
        flag[i] = 0;
    }

    /* Choose num items */
    for (i = 0; i < num; i++) {
        /* Choose one from the n-i remaining items */
        choice = rand() % (n-i) + 1;

        /* Locate the chosen item, which is the choice'(th)  remaining item */
        j = 0;
        counter = 0;
        while (counter < choice) {
            if (!flag[j]) {
                counter++;
            }
            if (counter < choice) {
                j++;
            }
        }

        /* Mark the chosen item as chosen */
        flag[j] = 1;
        sum += items[j];
    }

    free(flag);
    return sum;
}
```

End of exam