CMPT 125 D100, Fall 2025

Midterm Exam - SOLUTIONS
October 28, 2025

Name__________________________

SFU ID: |__|__|__|__|__|__|__|__|__|

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. Duration of the exam is 100 minutes.
2. Write your full name and SFU ID **clearly**.
3. This is a closed book exam, no calculators, cell phones, or any other material.
4. The exam consists of four (4) problems.
5. Write your answers in the provided space.
6. There is an extra page at the end of the exam. You may use it if needed.
7. **Explain all your answers.**
8. **Really, explain all your answers.**

Good luck!

Problem 1 [25 points]

a) [2 points each] Suppose you have a program in C named *program.c*.

1. What is the command in Linux to compile it into an executable file?

   Answer: gcc program.c

2. What is the default name of the compiled executable?

   Answer: a.out

3. How can you specify the name of the executable in the compilation line?

   Answer: gcc program.c -o specific_name

b) [7 points] Will the code below compile? If yes, what will be the result of the execution? If there are any errors/warnings/potential issues explain them.

```
#include <stdio.h>
enum colors {RED, GREEN, BLUE, YELLOW};

int fun(int* x, int* y) {  x points to a, y points to b
      long z = 4;
      *y = z;         sets b=4
      *x = YELLOW;    sets a=3   (YELLOW is 3)
      *y = 8;         sets b=8 it does not affect z
      return z;       returns 4
}

int main() {
      int a = RED, b = GREEN;  sets a=0, b=1
      int c = fun(&a, &b);     sets a=3, b=8 , and returns 4 [see comments above]
      printf("a = %d, b = %d, c = %d\n", a, b, c);
      return 0;
}
```

Answer: it will compile
The function will print: a = 3, b = 8, c = 4

c) [6 points] Will the code below compile? If yes, what will be the result of the execution? If there are any errors/warnings/potential issues explain them.

```c
#include <stdio.h>

int main() {
    char str[15] = {0,28,29,30,31,'5','6','7',0};
    int ind=1;
    while (str[ind])
      ind++;

    printf("%s\n", str+ind-2);
    return 0;
}
```

Answer: it will compile
Ind runs until reaches 0.
Now str+ind-2 points to ‘6’.
The function will print: 67

d) [6 points] Will the code below compile? If yes, what will be the result of the execution? If there are any errors/warnings/potential issues explain them.

```c
#include <stdio.h>

int* what() {
  int arr[3];
  for(int i=0;i<3;i++)
    arr[i] = i;
  return arr;
}

int main() {
  int* a = what();
  a[0] = 100;
  printf("a = [%d, %d, %d]\n", a[0], a[1], a[2]);
  return 0;
}
```

Answer: what tries to return a pointer to a local array. Behaviour is unpredictable

Problem 2 [25 points]

a) [15 points] Write a function that gets a string `str`, and a char `delim`, and returns the number of tokens in the string separated by `delim`.

For example,

- `count_tokens("abc-EFG--", '-')` *needs to return 2.*
- `count_tokens("++a+b+c+++", '+')` *needs to return 3.*
- `count_tokens("***", '*')` *needs to return 0.*
- `count_tokens("abcaa", '*')` *needs to return 1.*

```
int count_tokens(const char* str, char delim) {



  // IDEA: read the string char by char
  // increase count if str[i] is not delim, but str[i-1] is delim


  int count=0; // counts the number of tokens
  bool is_prev_delim = true; // checks if previous char was delim

  for (int i=0; str[i]!=0; i++) {
    if (str[i] == delim) {  // if the char is delim
      is_prev_delim = true; // update the flag
    }
    else { // char is not delim
      if (is_prev_delim) {
        count++;  // increase count
        is_prev_delim = false; // update the flag
      }
    }
  }
  return count;








}
```

b) [10 points] Write a function that gets two strings `str1` and `str2`, and checks if `str2` is a suffix `str1`. For full marks, your function needs to run in time O(strlen(str1)+strlen(str2)). For example,

- `is_suffix("abcd", "cd")` *needs to return* `true`.
- `is_suffix("cd", "Acd")` *needs to return* `false`.
- `is_suffix("123v787", "v787")` *needs to return* `true`.
- `is_suffix("xyz", "xyz")` *needs to return* `true`.

You may assume the standard libraries are included (e.g., *stdio.h*, *stdlib.h*, stdbool, *string.h*).

```
bool is_suffix(const char* str1, const char* str2) {






  int len1 = strlen(str1);
  int len2 = strlen(str2);

  if (len2 > len1)
    return false;

  for (int i=0; i<len2;i++)
    if (str1[len1-1-i] != str2[len2-1-i])
      return false;
// alternatively, you could use strcmp(str1+len2-len1,str2);


  return true;











}
```

Problem 3 [25 points]

a) [5 points] Give an example of an array of length 10, on which **InsertionSort** makes exactly four swaps in each of the last two iterations of the outer loop, and no other swaps.

Answer: [1,2,3,4,7,8,9,10,5,6]

b) [5 points] Write a function called `pancake_flip`, that gets an array `arr` and integer `k`, and reverses the order of the first `k` elements in `arr` . We call it a `k-flip`.
For example,

- If `arr=[1,2,3,4,5,6,7,8]` and `k=3`, then after executing the function `arr` becomes `[`**3,2,1**`,4,5,6,7,8]`.
- If `arr=[1,2,3,4,5,6,7,8]` and `k=6`, then after executing the function `arr` becomes `[`**6,5,4,3,2,1**`,7,8]`.

(Note that k=0 or k=1 do not affect `arr`.) You may assume that the length of `arr` is at least `k`.

```
void pancake_flip(int* arr, int k) {

  int tmp;
  for (int i=0; i<k/2; i++) {
      tmp = arr[i];
      arr[i] = arr[k-1-i];
      arr[k-1-i] = tmp;
  }

}
```

c) [15 points] Write a function `pancake_sort` that gets an array `A` and sorts it using a sequence of k-flips from the previous item. The function returns an array that represents the sequence of k-flips that sort `A`. For example,

- `pancake_sort(A =[3,2,4,1])` *can return the sequence* ***[5, 3,4,2,3,2]****.*
  The zeroth element is *the length of the sequence* *(in this case 5), followed by a sequence of k's.*
  By applying the 5 flips the result will be:
  3: `A =[4,2,3,1]`
  4: `A =[1,3,2,4]`
  2: `A =[3,1,2,4]`
  3: `A =[2,1,3,4]`
  2: `A =[1,2,3,4]`

Note: the answer is not unique. For example for `A =[3,2,4,1])` *the function can also return the sequence* ***[4, 4,2,4,3]*** *-- 4 is the length of the sequence, and the flips are [4,2,4,3].*

Explain your idea before writing code!

```
// The function gets an array A of length len.
// It returns an array SEQ specifying the sequence of k-flips,
// so that the following code sorts A:
// -----------------------
//     for (i=1; i <= SEQ[0]; i++)
//       pancake_flip(A,SEQ[i]);
// -----------------------
// That is, SEQ[0] holds the length of the sequence,
// and A is sorted by the sequence of flips SEQ[1…SEQ[0]].
// (Note: the returned array must be allocated on the heap)
int* pancake_sort(int* A, int len) {
```

IDEA: We can do something similar to Insertion Sort (in reverse order).
First find maximum in A, and send it to position A[len-1] using 2 flips. Specifically, suppose A[m] is maximal, then we apply:

- pancake_flip(m+1) sends max to A[0]
- pancake_flip(len) sens max to A[len-1]

Now max is in place, and we continue to the subarray A[0…len-2]. We find max in A[0…len-2], and then send it to A[len-2] using two flips the same way. If A[m] is maximum, we apply:

- pancake_flip(m+1) sends max to A[0]
- pancake_flip(len) sends max to A[len-2]

And so on… Total we will have at most 2*len flips.

```
    int seq_len = 0;
    int* ret = malloc((2*len + 1) * sizeof(int));

    for (int i=0; i<len-1;i++) {
        int m = find_max(arr, len-i);  // find max elt in A[0…len-1-i]
        ret[++seq_len] = m+1;
        ret[++seq_len] = len-i;
        pancake_flip(arr, m+1);       // put max in A[0]
        pancake_flip(arr, len-i);     // put max in A[len-1-i]
    }
    ret[0] = seq_len
    return ret;
}


int int find_max(int* arr, int len) { // finds index of max elt in arr
    int max_ind = 0;
    for (int i=1;i<len;i++)
        if (arr[i] > arr[max_ind])
            max_ind = i;
    return max_ind;
}
```

Problem 4 [25 points]

Consider the following function.

```
int foo(unsigned int n) {
  if (n<=3)
    return n;
  return 3*foo(n-1) + 2*foo(n-2) + 1*foo(n-3);
}
```

a) [5 points] Compute foo(5). Explain your answer.

Answer: base case: foo(1)=1, foo(2)=2, foo(3)=3,
Recursion:
foo(4)=3*foo(3)+2*foo(2)+1*foo(1)=3*3+2*2+1*1=14
foo(5)=3*foo(4)+2*foo(3)+1*foo(2)=3*14+2*3+1*2=50

b) [8 points] Rewrite the function foo() with the same functionality so that on input n, it returns the answer in time O(n). Explain your answer.

```
int foo(unsigned int n) {
  if (n<=3)
    return n;

  int* a = (int*)malloc((n+1)*sizeof(int));
  a[0]=0;   a[1]=1;   a[2]=2;   a[3]=3;

  for (int i=4; i<=n; i++)
    a[n] = 3*a[n-1] + 2*a[n-2] + a[n-3];

  int ret = a[n];
  free(a);
  return ret;
}
```

c) [12 points] Consider the following definition of a Linked List.

```
  struct LL_node {
    int data;
    struct LL_node* next;
  };
  typedef struct LL_node LL_node_t;

  typedef struct {
    LL_node_t* head;
  } LL_t;
```

Write a function that gets a Linked List and prints it in reverse order in time O(n), where n is the length of the list.

```
  void print_reverse(LL_t* list) {

    if (list==NULL)
      return;
    else
      print_reverse_helper(LL_t* list->head) {
  }


  // helper function that takes a head of the list as an argument
  void print_reverse_helper(LL_node_t* head) {
    if (head==NULL)   // base case
      return;
    else {
      print_reverse_helper(head->next) {
      printf("%d", head->data);
    }
  }
```

Extra page

Empty page