

CSC 111 - FALL 2021

FUNDAMENTALS OF PROGRAMMING WITH ENGINEERING APPLICATIONS

PRACTICE FINAL EXAMINATION

UNIVERSITY OF VICTORIA

Date: Fall 2021**CRN:** 00000**Instructor:** B. Bird

Student Name:	
Student Number:	
Signature:	

1. You have 180 minutes (3 hours) to complete this exam.
2. Answer all questions in the space provided on this exam paper.
3. The invigilators, at their discretion, may require you to hand in any of the notes or books you used during the exam when you hand in your test paper.
4. There are 11 questions on 15 pages, including this cover page. The last page is a reference sheet for the C standard library. Please verify that your copy has all pages and notify the invigilator immediately if any pages are missing.

Question [Max. Marks]	Mark
1 [7]	
2 [14]	
3 [5]	
4 [5]	
5 [7]	
6 [4]	

Question [Max. Marks]	Mark
7 [8]	
8 [5]	
9 [10]	
10 [5]	
11 [10]	
Total [80]:	

Question 1 [7 marks] Consider the syntactically correct C declarations below.

```

1 int a = 17;
2 int b = 10;
3 float x = 11.1;
4 float y = 11.6;
5 float* F[2] = {&x, &y};

```

In both of the parts below, each row of the table should be considered independent from the other rows, so if one expression modifies a variable's value, the modification does **not** persist for the other rows of the table.

(a) For each expression below, give the **type** of the result. If an expression is not syntactically correct, write 'ERROR' instead of giving a type.

Expression	Type
a	int
x/a + y/b	
*F[0] + *F[1]	

(b) For each expression below, give the **value** of the result. If an expression is not syntactically correct, write 'ERROR' instead of giving a value.

Expression	Value
a	17
(a+b)%(a-b)	
x + (a/b)	
(x+a)/b	
(a++) + x	
y + (--b)	

Question 2 [14 marks] Consider the syntactically correct C declarations below.

```

1 typedef struct {
2     int x;
3     int* z;
4 } MysteryStruct;
5 int A[5] = {6, 10, 17, 187, 111};
6 int B[3] = {3, 14, 15};
7 int* p = &A[1];
8 MysteryStruct M1, M2;
9 M1.x = -1;
10 M1.z = &B[0];
11 M2.x = 100;
12 M2.z = &A[0];
13 MysteryStruct* q = &M1;

```

In both of the parts below, each row of the table should be considered independent from the other rows, so if one expression modifies a variable's value, the modification does **not** persist for the other rows of the table.

(a) For each expression below, give the **type** of the result. If an expression is not syntactically correct, write 'ERROR' instead of giving a type.

Expression	Type	Expression	Type
*q		(*q)->z	
M1 + M2		*p + q->x	
*q->z		p + 1	

(b) For each expression below, give the **value** of the result. If an expression is not syntactically correct, write 'ERROR' instead of giving a value.

Expression	Type	Expression	Type
*(p + 1)		*q->z	
*M1.z		(*q).x	
q.x		*(&M1)->z++	
*q.z		*(M1 = M2).z	

Question 3 [5 marks] What is the output of the syntactically correct C program below?

```

1 #include <stdio.h>
2
3 int main() {
4     int n = 15;
5     int i;
6     for (i = 0; i < n; i++){
7         if (i % 2 == 0)
8             printf("%d ", i);
9         if (i % 5 == 1)
10            printf("%d ", i);
11        n--;
12    }
13    printf("\n");
14    return 0;
15 }
```

Answer:

Question 4 [5 marks] What is the output of the syntactically correct C program below?

```

1 #include <stdio.h>
2 #include <ctype.h>
3 #include <string.h>
4
5 int main() {
6     char str[] = "raspberry";
7     char c = str[1];
8     int i = 0;
9     while(i < strlen(str)){
10        printf("%c ", c);
11        c = str[i];
12        i++;
13        if ( c != 'r' || c != 'a' ){
14            i++;
15        }
16    }
17    printf("\n");
18    return 0;
19 }
```

Answer:

Question 5 [7 marks] Consider the syntactically correct C structure definition below.

```
1 typedef struct{
2     char name[1000];
3     float price;
4 } MenuItem;
```

(a) In the space below, write the definition for the function `new_menu_item` (with the function signature given) which takes the name and price of a menu item and returns a new `MenuItem` structure containing both values. You may assume that the `string.h` library header has been included.

```
MenuItem new_menu_item( char item_name[], float item_price ){
```

```
}
```

(b) In the space below, write the definition for the function `print_menu` (with the function signature given) which takes an array of `MenuItem` structures and prints each menu item in the format “Name - \$price” (for example, if a particular item has name “Caesar Salad” and price 6.10, the function will generate the line “Caesar Salad - \$6.10”).

```
void print_menu( int number_of_items, MenuItem all_items[number_of_items] ){
```

```
}
```

Question 6 [4 marks] What is the output of the syntactically correct C program below?

```
1  #include <stdio.h>
2
3  void a_function(char S1[], char S2[], char output[]) {
4      int j = 0;
5      int k = 0;
6      output[k] = S1[k];
7      k++;
8      while(S1[j] != '\0' && S2[j] != '\0'){
9          output[k] = S2[j];
10         k++;
11         j++;
12     }
13     output[k] = '\0';
14 }
15
16 int main(){
17     char string1[] = "foe";
18     char string2[] = "old";
19     char string3[] = "shoe";
20     char string4[] = "cold";
21     char string5[] = "boot";
22     char string6[] = "locked";
23
24     char output[100];
25
26     a_function(string1, string2, output);
27     printf("%s %s: %s\n",string1, string2, output);
28     a_function(string3, string4, output);
29     printf("%s %s: %s\n",string3, string4, output);
30     a_function(string5, string6, output);
31     printf("%s %s: %s\n",string5, string6, output);
32
33     return 0;
34 }
```

Question 7 [8 marks] Consider the syntactically correct C code below, which is missing a function `last_word`.

```

1  #include <stdio.h>
2  /* last_word(input_string, output_array)
3     Given a null terminated C string input_string and an output output_array,
4     set output_array to contain a null-terminated C string containing the last
5     word in input_string. You may assume that the input string contains only
6     letters (uppercase/lowercase) and spaces. No newlines, tabs, punctuation
7     or other characters will appear. The last word is defined to be the last
8     sequence of one or more contiguous non-space characters in the string.
9     See the examples below for various special cases.
10
11     Note that the output string should be empty if the input string is empty
12     or if the input string contains no word characters.
13     Don't forget the null terminator.
14
15     You may assume that output_array will always have enough space to store the
16     desired output (so no bounds checking is needed).
17 */
18 /* (your code would be placed here) */
19 int main(){
20     char S1[] = "Hello World";
21     char S2[] = " Raspberry";
22     char S3[] = "Programming";
23     char S4[] = " This string starts and ends with spaces ";
24
25     char output[100];
26     last_word(S1, output);
27     printf("Result for S1: \"%s\"\n", output );
28     last_word(S2, output);
29     printf("Result for S2: \"%s\"\n", output );
30     last_word(S3, output);
31     printf("Result for S3: \"%s\"\n", output );
32     last_word(S4, output);
33     printf("Result for S4: \"%s\"\n", output );
34     return 0;
35 }
```

Once the `last_word` function is implemented correctly, the program will generate the following output.

```

Result for S1: "World"
Result for S2: "Raspberry"
Result for S3: "Programming"
Result for S4: "spaces"
```

Write a definition of the function `last_word` (including the function signature). For full marks, your implementation should work correctly on all input values, not just the ones above.

```
void last_word (char in[], char out[]) {  
  
    int i;  
    int count=0;  
    int total = strlen(in)-1;  
    for (i=total;i>=0;i--) {  
        if ((in[i]==' ') && (i!=total)) {  
            break;  
        }  
        count++;  
    }  
    int left = total - count;  
  
    for (i=0;i<count+1;i++) {  
        if (in[left+i+1]==' ') {  
            break;  
        }  
        out[i]=in[left+i+1];  
    }  
    out[i] = '\0';  
}
```


Question 8 [5 marks] Consider the syntactically correct C code below, which is missing a function `count_less`.

```

1  #include <stdio.h>
2
3  /* count_less(n, T, v)
4     Given an n x n integer array T, along with a value v,
5     count and return the number of entries of T which are less
6     than or equal to v. */
7  /* (your code from below would be placed here) */
8  int main(){
9     int T1[3][3] = { { 10, 14, 2021},
10                     { 11, 8, 2021},
11                     {111, 116, 445} };
12     int T2[4][4] = { { 9,10,11,12 },
13                     { 5, 6, 7, 8 },
14                     { 1, 2, 3, 4 },
15                     {13,14,15,16 } };
16     printf("Elements less than 6 in T1: %d\n", count_less(3, T1, 6) );
17     printf("Elements less than 17 in T1: %d\n", count_less(3, T1, 17) );
18     printf("Elements less than 6 in T2: %d\n", count_less(4, T2, 6) );
19     printf("Elements less than 17 in T2: %d\n", count_less(4, T2, 17) );
20     return 0;
21 }
```

Once the `count_less` function is implemented correctly, the program will generate the following output.

```

Elements less than 6 in T1: 0
Elements less than 17 in T1: 4
Elements less than 6 in T2: 6
Elements less than 17 in T2: 16
```

Write a definition of the function `count_less` (including the function signature). For full marks, your implementation should work correctly on all input values, not just the ones above.

Question 9 [10 marks] Consider the text file `cities.txt` below. You may assume that there are no extra spaces at the beginning or end of each line of the file.

`cities.txt`

Victoria
Campbell River
Nanaimo
Squamish
Ladysmith
Port Alberni

(a) What is the output of the syntactically correct C program below?

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main(){
6      FILE* input_file = fopen("cities.txt", "r");
7
8      if (input_file == NULL){
9          printf("Error\n");
10         return 1;
11     }
12     char line[100];
13     int k = 1;
14     while( fgets(line, 100, input_file) != NULL ){
15         int line_length = strlen(line);
16         line[line_length/2] = '\\0';
17         line[k] = '\\0';
18         if (k%6 > 0)
19             printf("%s\\n", line);
20     }
21     return 0;
22 }
```

(b) Describe a situation in which the return value of `fopen` would equal `NULL` (resulting in the program on the previous page printing 'Error' and exiting).

(c) In the space below, provide an implementation of the function `count_uppercase_in_file` according to the following specification and function signature.

- The function takes a string containing a filename as a parameter.
- If the file cannot be opened for reading, the function returns -1.
- Otherwise, the function opens the file and returns the number of uppercase letters in the file.

You are advised to use `fgetc` to read characters from the file and `isupper` to test if a character is uppercase.

Example: If the function is called with the filename "`cities.txt`" and reads the text file shown on the previous page, the return value will be 8.

```
int count_uppercase_in_file(char filename[]){
```

```
}
```

Question 10 [5 marks] What is the output of the syntactically correct C program below?

```
1  #include <stdio.h>
2
3  void ready(int* x){
4      int y = *x;
5      if (y > 0){
6          int* x = &y;
7          y = (*x)-1;
8          *x = y;
9          printf("ready: %d %d\n", *x, y);
10     }
11 }
12 int set(int x, int y){
13     x = y++;
14     if (x < y){
15         printf("set: %d %d\n", x, y);
16         return x;
17     }
18     return y;
19 }
20 int go(int* x, int* y){
21     int z = *x;
22     ready(&z);
23     set(*x,*y);
24     printf("go: %d %d\n", *x, *y);
25     *x = 100;
26     *y = 106;
27     return 111;
28
29 }
30 int main(){
31     int x = 6;
32     int y = 10;
33     x = go(&x, &y);
34     printf("main: %d %d\n", x, y);
35
36     return 0;
37 }
```

Question 11 [10 marks] Consider the syntactically correct C code below.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  typedef struct ListNode{
4      int element;
5      struct ListNode* next;
6  } ListNode;
7  typedef struct{
8      ListNode* head;
9  } LinkedList;
10
11 /* remove_first(L)
12    Given a pointer to a LinkedList structure, remove the first element
13    from the list. Remember to deallocate the memory for the node. */
14 void remove_first(LinkedList* L);
15
16 /* get_last(L)
17    Given a pointer to a LinkedList structure, return the value
18    of the last element in the list. */
19 int get_last(LinkedList* L);
20
21 void print_list(LinkedList* L){
22     for(ListNode* node = L->head; node != NULL; node = node->next){
23         printf("%d ", node->element);
24     }
25     printf("\n");
26 }
27 void add_front(LinkedList* L, int new_element){
28     ListNode* new_node = malloc( sizeof(ListNode) );
29     new_node->element = new_element;
30     new_node->next = L->head;
31     L->head = new_node;
32 }
33 int main(){
34     LinkedList L1, L2;
35     L1.head = NULL; L2.head = NULL;
36     add_front(&L1, 187); add_front(&L1, 17); add_front(&L1, 10); add_front(&L1, 6);
37     add_front(&L2, 225); add_front(&L2, 116); add_front(&L2, 111);
38     print_list(&L1);
39     printf("Last element: %d\n", get_last(&L1) );
40     print_list(&L2);
41     printf("Last element: %d\n", get_last(&L2) );
42     printf("After removing first element:\n");
43     remove_first(&L1); print_list(&L1);
44     remove_first(&L2); print_list(&L2);
45     return 0;
46 }

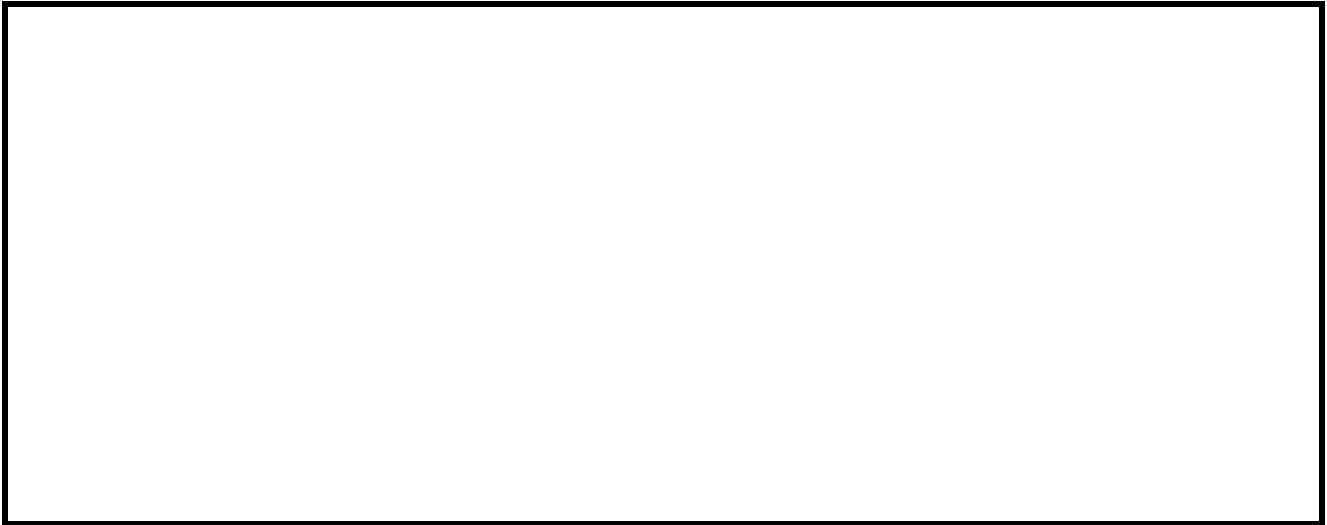
```

The functions `remove_first` and `get_last` are missing. Once they are implemented correctly, the program will generate the following output.

```
6 10 17 187
Last element: 187
111 116 225
Last element: 225
After removing first element:
10 17 187
116 225
```

In parts (a) and (b) below, your implementations work correctly on all lists, not just the specific example given above. You may assume that the list which is provided to your functions contains at least one element (so there is no need to check for an empty list).

(a) Write a definition of the function `remove_first` (including the function signature).



(b) Write a definition of the function `get_last` (including the function signature).



C Standard Library Reference (simplified)

Character Operations (ctype.h)

```
int isalpha(int c);
int isdigit(int c);
int islower(int c);
int isupper(int c);
int tolower(int c);
int toupper(int c);
```

General Functionality (stdlib.h)

```
void exit(int status);
void free(void* ptr);
void* malloc(unsigned int size);
int rand();
void srand(unsigned int seed);
```

Input and Output (stdio.h)

```
int fclose(FILE* f);
int feof(FILE* f);
int fgetc(FILE* f);
char* fgets(char buf[], int bufsize, FILE* f);
FILE* fopen(char filename[], char* mode);
int fputc(int character, FILE* f);
int fputs(char str[], FILE* f);
int fprintf(FILE* f, char format[], ...);
int fscanf(FILE* f, char format[], ...);
int printf(char format[], ...);
int scanf(char format[], ...);
```

Mathematics (math.h)

```
double cos(double x);
double exp(double x);
double log(double x);
double tan(double x);
double sin(double x);
double sqrt(double x);
```

Strings (string.h)

```
char* strcat(char destination[], char source[]);
int strcmp(char str1[], char str2[]);
char* strcpy(char destination[], char source[]);
int strlen(char str[]);
```