CSC 111 - FALL 2019
FUNDAMENTALS OF PROGRAMMING WITH ENGINEERING APPLICATIONS
PRACTICE MIDTERM 2
UNIVERSITY OF VICTORIA

Date: Fall 2019 **CRN**: 00000 **Instructor**: B. Bird

Student Name:	
Student Number:	
Signature:	

1. **You have 70 minutes to complete this exam.**
2. **There are 8 questions on 9 pages, including this cover page. Please verify that your copy has all pages and notify the invigilator immediately if any pages are missing.**

Question [Max. Marks]	**Mark**
1 [9]	
2 [6]	
3 [7]	
4 [5]	
5 [5]	
6 [3]	
7 [5]	
8 [5]	
Total [45]:	

Question 1 [9 marks] Consider the syntactically correct C declarations below.

```
int a = 6;
int b = 10;
int c = 17;
int* p = &a;
int* q = &b;
int* r = &c;
int** X = &p;
int** Y = &q;
int** Z = &r;
```

(a) For each of the syntactically correct C expressions in the table below, give the **type** of the result. Assume that each expression is independent (so any modifications made by one expression have no effect on the results of the other expressions in the table).

Expression	Type
`p`	`int*`
`*X`	
`*r`	
`**Y`	
`&Z`	
`**(&q)`	
`&r`	

(b) For each of the syntactically correct C expressions in the table below, give the **value** of the result. Assume that each expression is independent (so any modifications made by one expression have no effect on the results of the other expressions in the table).

Expression	Value
`a`	6
`*q + *r`	
`**Z - *r`	
`b + *q + **Y`	

Question 2 [6 marks] What is the output of the syntactically correct C program below?

```
#include <stdio.h>

int fiddle(int** one_pointer, int* another_pointer){
   int i = *another_pointer;
   **one_pointer = 0;
   *one_pointer = another_pointer;
   return i + 100;
}
int main(){
   int a = 6;
   int b = 10;
   int c = 17;
   int* p = &a;
   int* q = &b;
   int* r = &c;
   int** X = &p;
   int** Y = &q;
   int** Z = &r;

   printf("%d %d %d\n", *p, *q, *r );
   p = q;
   printf("%d %d %d\n", *p, *q, *r );
   **X = 111;
   printf("%d %d %d\n", *p, *q, *r );
   *Y = *Z;
   printf("%d %d %d\n", *p, *q, *r );
   b = fiddle(Z,p);
   q = &b;
   printf("%d %d %d\n", *p, *q, *r );
   a = fiddle(X,q);
   r = &a;
   printf("%d %d %d\n", *p, *q, *r );

   return 0;
}
```

Question 3 [7 marks] Consider the syntactically correct C declarations below.

```
int x = 6;
int y = 10;
int A[5] = {314, 159, 265, 358, 979};
int B[5] = {-1, -3, -5, -7, -9};
int* Z[5] = { &x, &A[2], &A[3], &B[2], &y };
```

(a) For each of the syntactically correct C expressions in the table below, give the **type** of the result. Assume that each expression is independent (so any modifications made by one expression have no effect on the results of the other expressions in the table).

Expression	**Type**
`x`	`int`
`A[2]`	
`&A[2]`	
`Z[4]`	

(b) For each of the syntactically correct C expressions in the table below, give the **value** of the result. Assume that each expression is independent (so any modifications made by one expression have no effect on the results of the other expressions in the table).

Expression	**Value**
`x`	6
`A[2]`	
`A[1] + B[1]`	
`*Z[1]`	

(c) Write **one** assignment statement which changes the value of variable `y` to be `999`, **without using the name of** `y`.

Answer:

Question 4 [5 marks] Consider the syntactically correct C code below, which is missing a function `copy_positive`.

```
#include <stdio.h>

/* copy_positive(A, Aout, size)
   Given an array A, which will have the provided size, copy all positive
   (non-negative/non-zero) elements of A into the provided output array Aout.
   Return the size of the resulting array. */
/* (your code from below would be placed here) */

void print_array(int arr[], int n){
   for( int i = 0; i < n; i++ )
      printf("%d ", arr[i]);
   printf("\n");
}
int main(){
   int A1[] = {0, 9, -1, 0, 6, 10, 17};
   int A2[] = {11, 1, 0, -5};
   int B[100];  int b_size;
   b_size = copy_positive(A1,B,7);
   print_array(B, b_size);
   b_size = copy_positive(A2,B,4);
   print_array(B, b_size);
   return 0;
}
```

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

```
9 6 10 17
11 1
```

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

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

```c
#include <stdio.h>

typedef int Table[100][100];

/* print_diagonal(T, n)
   Given a Table T, which will have n rows and n columns, print all of the
   entries on the main diagonal of T (that is, entries whose row number and
   column number are equal). Remember to print a newline at the end. */
/* (your code from below would be placed here) */
int main(){
   Table T1 = { { 10, 10, 2019},
                {  11, 14, 2019},
                {111,116, 225} };
   Table T2 = { { 1, 2, 3, 4 },
                { 5, 6, 7, 8 },
                { 9,10,11,12 },
                {13,14,15,16 } };
   print_diagonal(T1, 3);
   print_diagonal(T2, 4);
   return 0;
}
```

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

```
10 14 225
1 6 11 16
```

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

Question 6 [3 marks] What is the output of the syntactically correct C program below?

```
#include <stdio.h>

void a_function(char S1[], char S2[], char output[]){
   int j, k;
   j = 0;
   k = 0;
   while(S1[j] != '\0' && S2[j] != '\0'){
      output[k] = S1[j];
      k++;
      output[k] = S2[j];
      k++;
      j++;
   }
   output[k] = '\0';
}

int main(){
   char string1[] = "foe";
   char string2[] = "old";
   char string3[] = "shoe";
   char string4[] = "cold";

   char output[100];

   a_function(string1, string2, output);
   printf("%s %s: %s\n",string1, string2, output);

   a_function(string3, string4, output);
   printf("%s %s: %s\n",string3, string4, output);

   return 0;
}
```

Question 7 [5 marks] Consider the syntactically correct C code below, which is missing a function `shift_string_right`.

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

/* (your code from below would be placed here) */

int main(){
   char s1[] = "earth";
   char s2[] = "electives";
   printf("First example: %s ", s1);
   shift_string_right(s1);
   printf("%s\n", s1);

   printf("Second example: %s ", s2);
   shift_string_right(s2);
   printf("%s\n", s2);
   return 0;
}
```

In the space below, write the definition of the `shift_string_right` function which is missing from the code above. The `shift_string_right` function will take a null-terminated C string and shift each character one position to the right. The character at the end of the string is moved to the empty space created at the beginning. Hint: Making a second copy of the string might make things easier.

When your code is correct, the program above will produce the following output.

```
First example: earth heart
Second example: electives selective
```

```
void shift_string_right(char s[]){

}
```

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

```
#include <stdio.h>

/* strings_equal(str1, str2)
   Given two strings, return 1 if they are equal (have the same length and
   contain the same sequence of characters) and 0 otherwise.
   */
/* (your code from below would be placed here) */

int main(){
   char S1[] = "Hello World";
   char S2[] = "Hello World";
   char S3[] = "Hello";
   char S4[] = "Raspberry Jam";
   char S5[] = "Blueberry Pie";

   printf("strings_equal(S1, S2): %d\n", strings_equal(S1, S2) );
   printf("strings_equal(S1, S3): %d\n", strings_equal(S1, S3) );
   printf("strings_equal(S4, S5): %d\n", strings_equal(S4, S5) );
   return 0;
}
```

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

```
strings_equal(S1, S2): 1
strings_equal(S1, S3): 0
strings_equal(S4, S5): 0
```

Write a definition of the function `strings_equal` (**including the function signature**). For full marks, your implementation should work correctly on all input values, not just the ones above. You may use any features of the standard library **except for** the `strcmp` function and its variants.