CMPT 130: Lab Work Week 11

1. Analyze the following program and determine its output

```
int main()
{
    char ch = 'A';
    char *p = new char(ch);
    char *q;
    q = &ch;
    char r[] = {ch};

    cout << ch << endl;
    cout << p << endl;
    cout << q << endl;
    cout << r << endl;
    cout << &ch << endl;
    cout << &(p[0]) << endl;
    cout << &(q[0]) << endl;
    cout << &(r[0]) << endl;

    cout << &(*p) << endl;
    cout << &(*q) << endl;
    cout << &(*r) << endl;

    delete[] p;

    system("Pause");
    return 0;
}
```

Here all we need to observe is that all the statements

```
cout << &(ch) << endl;
cout << &(p[0]) << endl;
cout << &(q[0]) << endl;
cout << &(r[0]) << endl;
cout << &(*p) << endl;
cout << &(*q) << endl;
cout << &(*r) << endl;
```

will attempt to print a value of a pointer to char data type. Thus all these statements will be equivalent to printing the value of a pointer to char data type such as

```
cout << p << endl;
```

2. It should be emphasized very well that C++ string is totally different from C-string. A C++ string is a data type that requires **#include <string>** and supports many operations such as indexing, default or non-default initialization, length built-in function, concatenation, etc... But a C-string is simply an array of characters with a null character as its last element and supports only indexing, non-default initialization etc,... Analyze the following program and determine its output. Also identify the statements that have syntax or semantic or run –time errors and correct them as you see fit best.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
        //Let's play with C++ strings
        string s1;
        string s2 = "Yonas ";
        s1 = "Instructor";
        string s3 = s1 + s2;
        cout << s3 << endl;

        //Now let's play with C-strings
        char *C1;
        char C2[] = "Yonas ";
        C1 = "Instructor";
        char *C3 = C1 + C2;
        cout << C3 << endl;

        system("Pause");
        return 0;
}
```

3. As you can see in Q2 above, given two C-strings C1 and C2, the operation C1+C2 is syntax error. That is, C-strings cannot be concatenated using the + operator. In this question, we are going to write our own function named **`combine`** that takes two C-string arguments and returns a new C-string which is made up of all the characters of C1 followed by those of C2. Consider the following C++ main program and make it work by providing the definition of the function **`combine`**.

```
int main()
{
    //Create a C-string
    const int len1 = rand() % 10 + 4;
    char *C1 = new char[len1 + 1];
    for (int i = 0; i < len1; i++)
           C1[i] = rand() % 26 + 97;
    C1[len1] = '\0';

    //Create a second C-string
    const int len2 = rand() % 10 + 6;
    char *C2 = new char[len2 + 1];
    for (int i = 0; i < len2; i++)
           C2[i] = rand() % 26 + 65;
    C2[len2] = '\0';

    //Print the two C-strings
    cout << "C1 = " << C1 << endl;
    cout << "C2 = " << C2 << endl;

    //Combine C1 and C2 to a new C-string C3
    char *C3 = combine(C1, C2);

    //Print the answer
    cout << C1 << " and " << C2 << " combined together give " << C3 << endl;

    //Delete the dynamic arrays
    delete[] C1; delete[] C2; delete[] C3;

    system("Pause");
    return 0;
}
```

Here is a sample output to help you test your program.

```
C1 = hqghu
C2 = EAYLNLFDXFIRCV
hqghu and EAYLNLFDXFIRCV combined together give hqghuEAYLNLFDXFIRCV
Press any key to continue . . .
```

4. Write a function named **findCharacter** that takes a C-string and a character as arguments and returns the first index of the character in the C-string if the character is found in the C-string; otherwise returns -1.

For example, if **char S[ ] = "massachussettes";** and **char ch1 = 's', ch2 = 'c';** then

- The function call **findCharacter(S, ch1)** must return **2**.
- The function call **findCharacter(S, ch2)** must return **5**.

5. Write a function named **findLastCharacter** that takes a C-string and a character as arguments and returns the last index of the character in the C-string if the character is found in the C-string; otherwise returns -1.

For example, if **char S[ ] = "massachussettes";** and **char ch1 = 's', ch2 = 'c';** then

- The function call **findLastCharacter(S, ch1)** must return **14**.
- The function call **findLastCharacter(S, ch2)** must return **5**.

6. Write a function named **doesContainDigit** that takes a C-string as an argument and returns true if the C-string contains a digit and returns false otherwise.

For example, if char **S1[ ] = "test";** and **char S2[ ] = "cmpt 120";** then
- The function call **doesContainDigit(S1)** must return **false**.
- The function call **doesContainDigit(S2)** must return **true**.

7. Write a function named **doesContainLower** that takes a C-string as an argument and returns true if the C-string contains a lower case English alphabet and returns false otherwise.

8. Write a function named **doesContainUpper** that takes a C-string as an argument and returns true if the C-string contains an upper case English alphabet and returns false otherwise.

9. Write a function named **doesContainAlphabet** that takes a C-string as an argument and returns true if the C-string contains an English alphabet (that is either a lower or upper case English alphabet) and returns false otherwise.

10. Write a function named **isLower** that takes a C-string as an argument and returns true if all the printable characters of the C-string are lower case English alphabets and returns false otherwise.

11. Write a function named **isUpper** that takes a C-string as an argument and returns true if all the printable characters of the C-string are upper case English alphabets and returns false otherwise.

12. Write a function named **isNumber** that takes a C-string as an argument and returns true if all the printable characters of the C-string are digits and returns false otherwise.

13. Write a function named **isAlphabet** that takes a C-string as an argument and returns true if all the printable characters of the C-string are English alphabets (that is either lower or upper case English alphabets) and returns false otherwise.

14. Write a C++ function named **countLowerAlphabets** that takes a C-string and returns the number of lower alphabet characters in the C-string.

15. Write a C++ function named **getLowerAlphabets** that takes a C-string argument and returns a C-string made up of the lower alphabet characters of the argument. If none of the characters of the argument is a lower alphabet then you must still return a C-string which is obviously empty C-string; that is a C-string with only the null character.

For example, the following code snapshot must print the C-string "mall"

```
char S1[ ] = "CmPT 120 Fall";
char* S2 = getLowerAlphabets(S1);
cout << S2 << endl;
```

Remark:- **(i)** Make sure to use dynamic array in order to return a C-string (i.e. dynamic array of character) from your function; **(ii)** Use the function **countLowerAlphabets** you wrote above to help you find out how big array you need to create; and **(iii)** Remember your argument is a C-string and you are required to return a C-string; which means both MUST have a null character at the end.

16. Write a C++ function named **countUpperAlphabets** that takes a C-string and returns the number of upper alphabet characters in the C-string.

17. Write a C++ function named **getUpperAlphabets** that takes a C-string argument and returns a C-string made up of the upper alphabet characters of the argument.

18. Write a C++ function named **countDigits** that takes a C-string and returns the number of digit characters in the C-string.

19. Write a C++ function named **getDigits** that takes a C-string argument and returns a C-string made up of the digit characters of the argument.

20. Write a C++ function named **countAlphabets** that takes a C-string and returns the number of alphabet characters in the C-string.

21. Write a C++ function named **getAlphabets** that takes a C-string argument and returns a C-string made up of the alphabet characters of the argument.

22. Recall that a given string is palindrome if it reads the same forward and backward. Example: **RACECAR** is a palindrome. Write a function named **isPalindrome** that takes a C-string as an argument and returns true if the printable characters of the C-string form a palindrome. Otherwise returns false.

23. Recall that a given string is an English language pangram if it contains all the English alphabets. For example **the quick brown fox jumps over the lazy dog** is a pangram string. Write a function named **isPangram** that takes a C-string as argument and returns true if the printable characters of the C-string form an English-language pangram. Otherwise returns false. Assume the C-string argument contains lower case English letters only.

24. Write a function named **countCharacter** that takes a C-string and a character as arguments and returns the number of the printable characters of the C-string argument that are equal to the character argument.

25. Write a function named **countCharacters** that takes two C-strings C1 and C2 as arguments and returns the number of characters of C1 that are also found in C2.

26. Write a function named **isContained** that takes two C-strings C1 and C2 and returns true if all the printable characters of C1 are contained in C2; return false otherwise. Hint:- Use your function **countCharacters** defined above.

27. Write a C++ function named **countCharactersUpTo** that takes a C-string **S**, a character **ch**, and an integer **k** as arguments and returns the number of times the character **ch** is found in the C-string **S** starting from index **0** (inclusive)up to index **k** (inclusive).

28. Write a C++ function named **countDistinctChars** that takes a C-string argument and returns the number of distinct characters in the C-string. For example if the C-string argument is "massachussettes", then your function must return 8 because there are 8 distinct characters in the C-string argument, namely m,a,s,c,h,u,e, and t. Hint:- Use your function **countCharactersUpTo** defined above**.**

29. Write a C++ function named **getDistinctChars** that takes a C-string argument and returns a C-string containing the distinct characters in the C-string argument. For example, if the C-string argument is

"massachussettes", then your function must return the C-string "maschuet". Hint:- Use your function **countDistinctChars** function.

30. Write a C++ function named **stringCompare** that takes two C-strings s1 and s2; and returns an integer with the following rule: return 1 if s1 is greater than s2; return 0 if s1 and s2 are equal; and return -1 if s1 is less than s2.

Remark:- Two C-strings are equal if they have the same length and characters at corresponding indexes are identical. A given C-string s1 is greater than another C-string s2 if the first distinguishing character of s1 is greater than that of s2 according to ascii code. Similarly for less than.

31. Write a main program suitable to test your function **stringCompare**

32. Write a C++ function named **stringCopy** that takes two C-strings s1 and s2 and copies the characters of s1 to s2. Note that when we call this function from the main program, both S1 and S2 are created such that the null character is placed at the end of both of them. The copying of characters of S1 to S2 must follow the following rules:

Rule 1: If the length of S1 and S2 are equal, then copy all the characters of S1 to S2.

Rule 2: If the length of S2 is smaller than length of S1 then S2 does not have enough space to copy all the characters of S1. Therefore this function should NOT copy any character of S1 to S2 and must return without doing anything.

Rule 3: If the length of S2 is greater than length of S1 then copy all the characters of S1 to S2 and put a null character next to the last character copied. Therefore in this case S2 will have two null characters: one at the end of it which was put in the main program and another next to the last character copied which is put in this function.

The main program to test the stringCopy function is given below.

```
int main()
{
	char S1[] = "BLABLABLABLA";
	int len = stringLength(S1);
	cout << "The C-string S1 is " << S1 << endl;
	cout << "The length of S1 is " << len << endl << endl;

	cout << "TEST 1. Copy S1 to S2 such that length of S2 is smaller than length of S1" << endl << endl;
	//Create a C-String S2 which is shorter than S1.
	//Initialize S2 with any characters
	//Put a null character at the end of S2
	char *S2 = new char[3];
	for (int i = 0; i < 2; i++)
		S2[i] = 'A';
	S2[2] = '\0';
	cout << "Originally, S2 is " << S2 << " and its length is " << stringLength(S2) << endl;
	//Copy S1 to S2
	stringCopy(S2, S1); //This function call will not copy anything from S1 to S2
	cout << "After stringCopy, the C-string S2 is " << S2 << endl;
	cout << "The length of S2 is " << stringLength(S2) << endl << endl;

	//Delete S2
	delete[] S2;
```

```
	cout << "TEST 2. Copy S1 to S2 such that length of S2 is exactly equal to length of S1" << endl <<
endl;

	//Create a C-String S2 which is same length as S1.
	//Initialize S2 with any characters
	//Put a null character at the end of S2
	S2 = new char[len+1]; //You need one more space for the \0
	for (int i = 0; i < len; i++)
		S2[i] = 'A';
	S2[len] = '\0';
	cout << "Originally, S2 is " << S2 << " and its length is " << stringLength(S2) << endl;
	//Copy S1 to S2
	stringCopy(S2, S1); //This function call will copy S1 to S2
	cout << "After stringCopy, the C-string S2 is " << S2 << endl;
	cout << "The length of S2 is " << stringLength(S2) << endl << endl;

	//Delete S2
	delete[] S2;

	cout << "TEST 3. Copy S1 to S2 such that length of S2 is bigger than length of S1" << endl << endl;

	//Create a C-String S2 which is much longer than S1.
	//Initialize S2 with any characters
	//Put a null character at the end of S2
	int s2Length = len+10;
	S2 = new char[s2Length+1];   //You need one more space for the \0
	for (int i = 0; i < s2Length; i++)
		S2[i] = 'A';
	S2[s2Length] = '\0';
	cout << "Originally, S2 is " << S2 << " and its length is " << stringLength(S2) << endl;
	//Copy S1 to S2
	stringCopy(S2, S1); //This function call will copy S1 to S2 and put \0 after the last character
copied
	cout << "After stringCopy, the C-string S2 is " << S2 << endl;
	cout << "The length of S2 is " << stringLength(S2) << endl;

	//Delete S2
	delete[] S2;

	system("Pause");
	return 0;
}
```

The output of this program with the correct implementation of stringCopy function is shown below: