Started on	Tuesday, September 28, 2021, 12:40 PM
State	Finished
Completed on	Tuesday, September 28, 2021, 3:26 PM
Time taken	2 hours 46 mins

Question **1**

Complete

Points out of 10.00

Write a function **`dbl_avg`** will take an **`int`** value as its first defined parameter, a pointer to a **`double`** as its second defined parameter, followed by a sequence of double-precision floating-point values whose count is equal to the value of the first defined argument. The function will compute the average of the sequence of double-precision floating-point values and assign the computed average to the object pointed to by the second defined parameter. The function will not return nothing. If the first defined parameter has value **`0`**, the function will compute an average value of **`0`**.

Make sure to include all necessary standard library header files to compile the program.

```c
#include <stdarg.h>
void dbl_avg (int value, double * dd, ...)
{
    if (!value)
    {
        *dd = 0;
        return;
    }
    va_list args;
    va_start(args, dd);

    int i = 0;
    while (i < value)
    {
        *dd += va_arg(args, double);
        i++;
    }

    va_end(args);
    *dd /= value;
}
```

Question **2**

Complete

Points out of 20.00

Define function **`min_printf`** that will process the format string for **format specifiers** with the following specifier characters **`%`**, **`c`**, **`d`**, **`x`**, **`f`**, and **`s`**. You should deal with these format specifiers similar to function **`printf`**. Just like **`printf`**, **`min_printf`** should return the number of characters written to **`stdout`**. **This document** describes use cases that should guide your definition of function **`min_printf`**.

```cpp
#include <stdarg.h>
#include <iostream>
#include <cstring>
#include <sstream>

int min_printf(const char *format, ...)
{
    int count = 0;
    std::string str;
    va_list args;
    va_start(args, format);
    while (*format)
    {
        char value = *format;
        if (*format == '%')
        {
            format++;
            value = *format;

            switch (value)
            {
                case 'c':
                {
                    char result = (char) va_arg(args, int);
                    str += result;
                    count++;
                    break;
                }

                case 'd':
                {
                    std::string result = std::to_string(va_arg(args, int));
                    for (const char c : result)
                    {
                        str += c;
                        count++;
                    }
                    break;
                }

                case 'x':
                {
                    std::stringstream ss;
                    ss << std::hex << va_arg(args, int);
                    for (const char c : ss.str())
                    {
                        str += c;
                        count++;
                    }
                    break;
                }

                case 'f':
                {
                    std::string result = std::to_string(va_arg(args,double));
                    for (const char c : result)
                    {
                        str+= c;
                        count++;
                    }
                    break;
                }

                case 's':
                {
                    const char *result = va_arg(args, const char*);
                    str += result;
                    int size = (int)strlen(result);
                    count += size;
                    break;
                }

                default:
                {
                    str += value;
                    count++;
                }
            }
        }
        else
        {
            str += value;
            count++;
        }
        format++;
    }
    std::cout << str;
    va_end(args);
    return count;
}
```

Question **3**

Complete

Points out of 10.00

Function **`int_adder`** will take an **`int`** value as its first and only defined parameter followed by a sequence of **`int`** values whose count is equal to the value of the defined parameter. The function will return the sum of the sequence of **`int`** parameters not including the defined parameter. If the defined parameter has value **`0`**, the function will return **`0`**.

Include all necessary standard library headers required to compile the function.

```
#include <iostream>
#include <stdarg.h>

int int_adder (int value, ...)
{
    if (!value)
    {
        return 0;
    }
    va_list args;
    va_start(args, value);

    int result = 0;
    int i = 0;
    while(i < value)
    {
        result += va_arg(args, int);
        i++;
    }

    va_end(args);

    return result;

}
```

Question **4**

Complete

Points out of 10.00

Define a function **`str_concat`** that will take a single defined parameter **`dst`** of type **`char*`** and an unknown number of parameters of type **`char const*`** and return a value of type **`char*`**.

The function appends to defined parameter **`dst`** copies of character strings that are pointed to by pointers in the unnamed argument list. The null terminating character in **`dst`** is overwritten by the first character of the first string in the unnamed parameter list, and a null character is included at the end of the new character string formed by the concatenation of both in destination. This is repeated for each character string pointed to by the pointers in the unnamed parameter list until a sentinel value of **`(char const*)0`** is reached. It is the responsibility of the user to ensure that the **`char`** buffer whose first element is pointed to by **`dst`** has sufficient size to contain the resultant concatenated string.

The return value is the address of the first element of the resultant concatenated string.

```
#include <stdarg.h>
#include <cstring>

char *str_concat(char* dst, ...)
{
    va_list args;
    va_start (args, dst);

    while (args)
    {
        const char *total = va_arg(args, const char*);
        if (!total)
            break;

        unsigned int size = strlen(dst);

        while (*total)
        {
            dst[size] = *total;
            total++;
            size++;
        }
        dst[size] = '\0';
        size++;
    }
    va_end(args);
    return dst;
}
```