

DOCUMENTATION OF “console_formatting.h”

ABSTRACTION(OR DEVELOPED) BY: MOHAMMED MAAZ BIN
KHAWAR

If you find any bug or just want to contribute in this open source
project just contact at: maazproductions25@gmail.com or
03162627726

1. dateAndTime (class)

It is a very simple class based originally on standard "ctime" library. The unusual thing about this class is that it is designed only to display current date and time only.

It contains no constructor. The member functions are as follows:

- **year()**

returns an integer equal to the current year. Takes no argument. Example:

```
dateAndTime now;  
cout << now.year();
```

OUTPUT:
2017

- **month()**

returns a string containing the first three alphabets of the current month, where first alphabet of each month is capitalized. Takes no argument. Example:

```
dateAndTime now;  
cout << now.month();
```

OUTPUT:
Jan

- **weekDay()**

returns a string value containing the full name of the current week day with first alphabet capitalized. Takes no argument. Example:

```
dateAndTime now;  
cout << now.weekDay();
```

OUTPUT:

Friday

- **day()**

returns an int equal to the current day of the current month. Range is 1 to 31. Takes no argument.

Example:

```
dateAndTime now;  
cout << now.day();
```

OUTPUT:

20

- **hour()**

returns an int equal to the current hour of the day. The format is 24-hour, so the range is 0 to 23.

Takes no argument. Example:

```
dateAndTime now;  
cout << now.hour();
```

OUTPUT:

5

- **minute()**

returns an int equal to the current minute of the current hour. Range is 0 to 59. Takes no argument.

Example:

```
dateAndTime now;  
cout << now.minute();
```

OUTPUT:

11

- **second()**

returns an int equal to the current second of the current minute. Range is 0 to 59. Takes no argument.

Example:

```
dateAndTime now;  
cout << now.second();
```

OUTPUT:

12

- **time12()**

returns a string value, containing a proper formatted time in 12-hour system. the format is hh:mm AM/PM. Takes no argument. Example:

```
dateAndTime now;  
cout << now.time12();
```

OUTPUT:

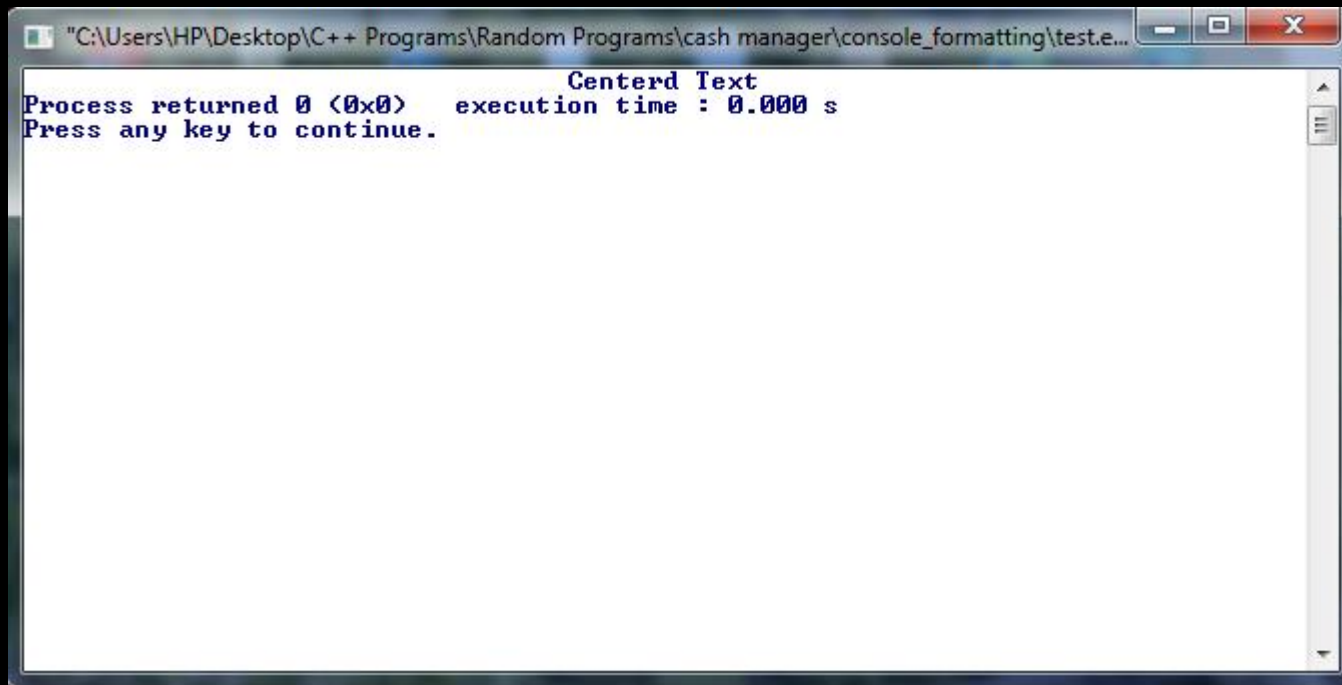
5:15 AM

2. center()

This function is made to automatically calculate and add the appropriate amount of "whitespaces" to the given text, so that it is placed right in horizontal center of the screen, irrespective of the length of that given text. It takes a string values as an argument and returns a string value having appropriate spaces attached before the original text. The function is best used as a stream manipulator. Example:

```
cout << center("Centerd Text"); /*"Centered Text" will be placed at the horizontal center of the console screen irrespective of it's length. */
```

OUTPUT:

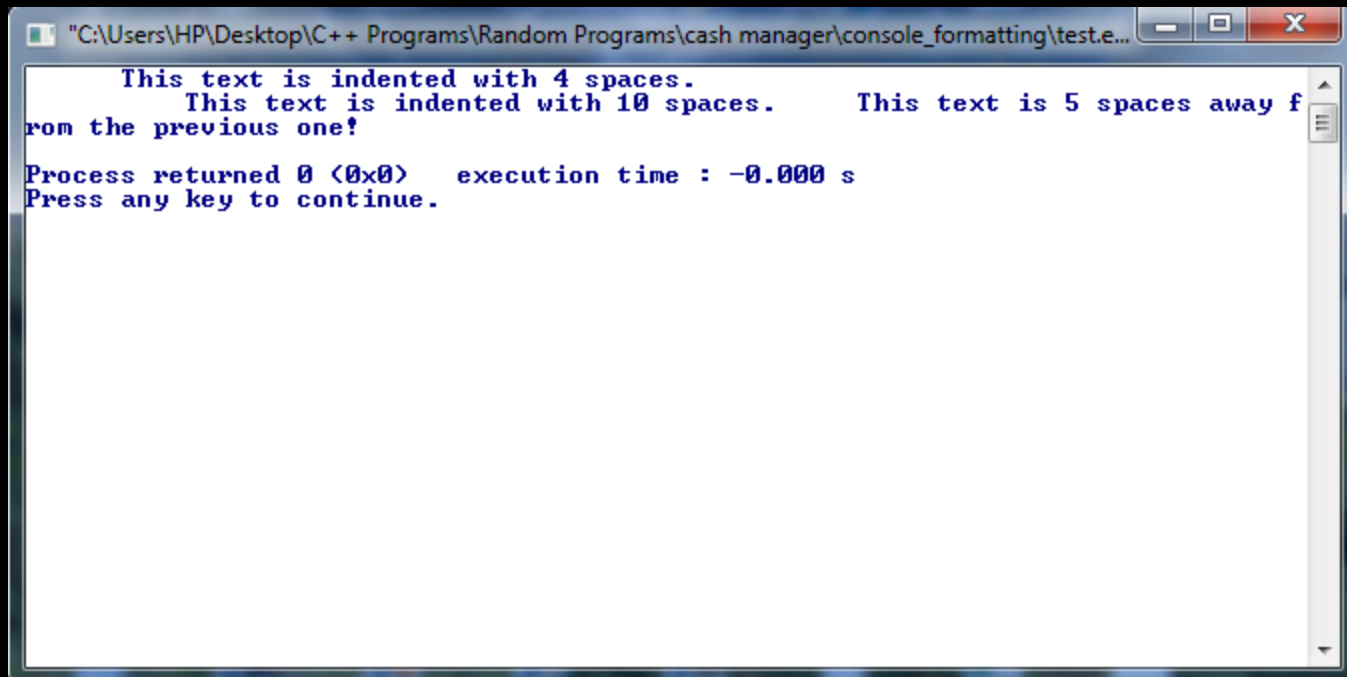
A screenshot of a Windows console window. The title bar shows the file path: "C:\Users\HP\Desktop\C++ Programs\Random Programs\cash manager\console_formatting\test.e...". The console output is as follows:
Centerd Text
Process returned 0 (0x0) execution time : 0.000 s
Press any key to continue.
The text "Centerd Text" is centered on the first line. The second line shows the process return code and execution time. The third line is a prompt to press any key to continue.

3. whiteSp()

This function adds given amount of spaces to the console starting from the cursor position. It takes a short int value as an argument which specifies the number of spaces to add. The function returns a string value just containing the number of spaces specified in the argument. The function is best used as a stream manipulator, and is handy for quick indenting purposes. Example:

```
cout << whiteSp(6) << "This text is indented with 4 spaces." << endl;  
cout << whiteSp(10) << "This text is indented with 10 spaces." << whiteSp(5) << "This text is 5 spaces away from the  
previous one!" << endl;
```

OUTPUT:



```
"C:\Users\HP\Desktop\C++ Programs\Random Programs\cash manager\console_formatting\test.e...  
This text is indented with 4 spaces.  
This text is indented with 10 spaces. This text is 5 spaces away f  
rom the previous one!  
Process returned 0 (0x0) execution time : -0.000 s  
Press any key to continue.
```

4. validateInt()

It is one of the most useful functions in this header file. This function can be used where an int value is required as an input from the user. The function is designed so the user is forced to enter an int value even if he tries to deliberately enter a non-int value. You can give a Boolean value as an argument to trigger the beep on or off, but it is optional and by default it is true. The function only reacts to numeral keys, minus sign key, backspace key, escape key, and enter key. If a key is pressed except these, then the function does not respond and instead gives an alert beep (if this trigger is not switched off initially). The detailed working is shown below:

Key Pressed	When accepted by the function	When rejected by the function
Numeral keys(0-9)	if the no. of digits are less than 11	when the no. of digits > 10
- (minus sign)	if pressed at the beginning	if pressed other than the beginning
Backspace key	if used when there is some number already written	if no number has been written yet
Escape Key	anytime during the user prompt	never
Return/Enter Key	if not used at the beginning	if used at the beginning, or the number written is beyond max-int value.

***NOTE:** by beginning I mean that the user has not written any number yet.

In this way the function always RETURNS an INT VALUE. This int value is the number user enters when asked for the prompt. The function also ensures that the user input must be within the range of the maximum int value. So if the user enters a number beyond this value he will not be able to enter this.

***Important Note:** The validateInt() function gives the user a previlage to just skip the number entering mode through escape key. For this purpose, when the user presses ESC key, the function returns the max-int value, i.e: 2147483647. Therefore there is a slight difference between standard int range and the range of this function, i.e in std int case the user can enter 2147483647 but values beyond this are not allowed. However in this case the user cannot enter 2147483647 and values beyond this. This is just to differentiate between the ESC key and the numeral keys. For demonstration, see the example below:

```
int val = validateInt();
if(val==2147483647)
{
    cout << endl << "Escape Key pressed! User input process terminated.";
}
else
{
    cout << endl << "The number you entered is " << val << ", and it is definitely an integer.";
}
```

If you still have any querries regarding this function, feel free to ask me at

maazproductions25@gmail.com

5. setBgClr() (only for windows)

This function sets the background color of the console. It takes a string argument as the name of the background color. The valid color names are:

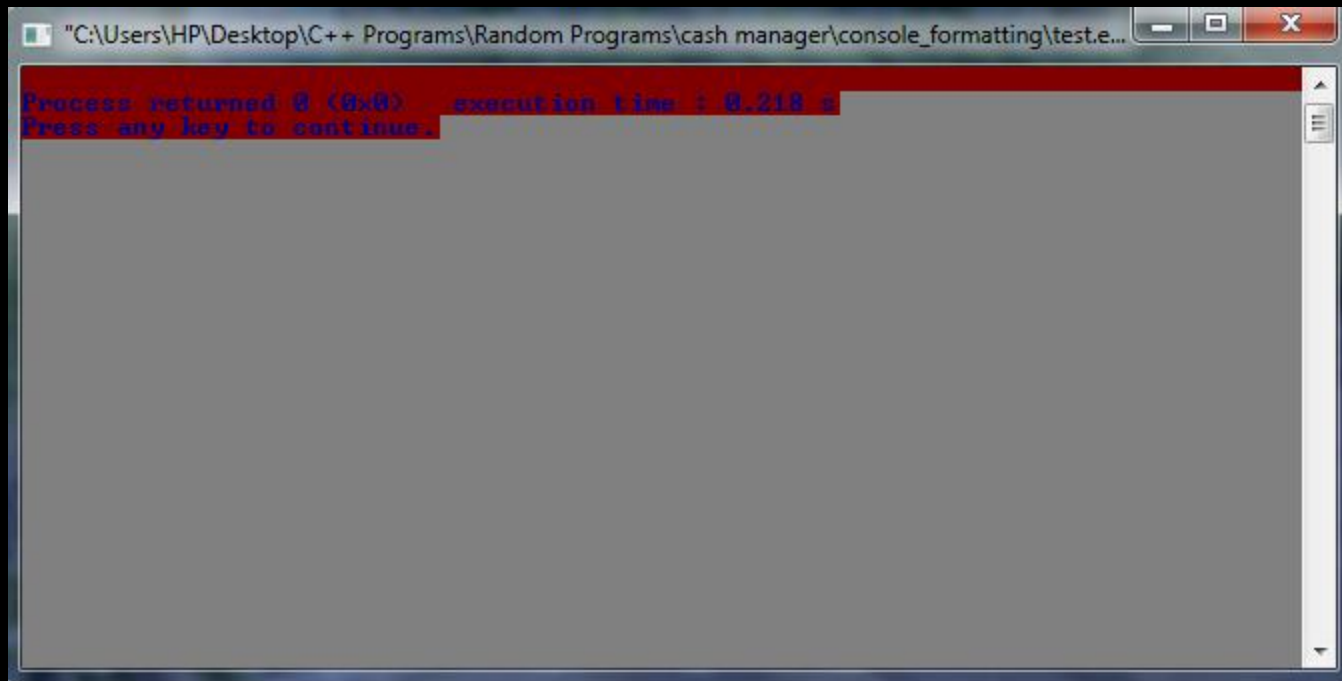
"Black", "Blue", "Green", "Cyan", "Red", "Purple", "Yellow", "White", "Gray", "L Blue", "L Green", "L Cyan", "L Red", "L Purple", "L Yellow", "Bright White"

You can also give a second optional int argument specifying the number of units of the console background to be colored. The first unit corresponds to top left unit. Each unit is equal in size of a character. Unit counting is done row wise. The function returns a Boolean value true if a valid color name is used. If the background color name does not match with any of the given valid names the current background color is restored and the function returns a Boolean false value. Example:

```
setBgClr("Gray"); //console background is set to Gray
```

```
setBgClr("Red",80); //The background-color of first row is set to red, as the length of one row of a console is usually 80 units
```

OUTPUT:



6. setFgClr() (only for windows)

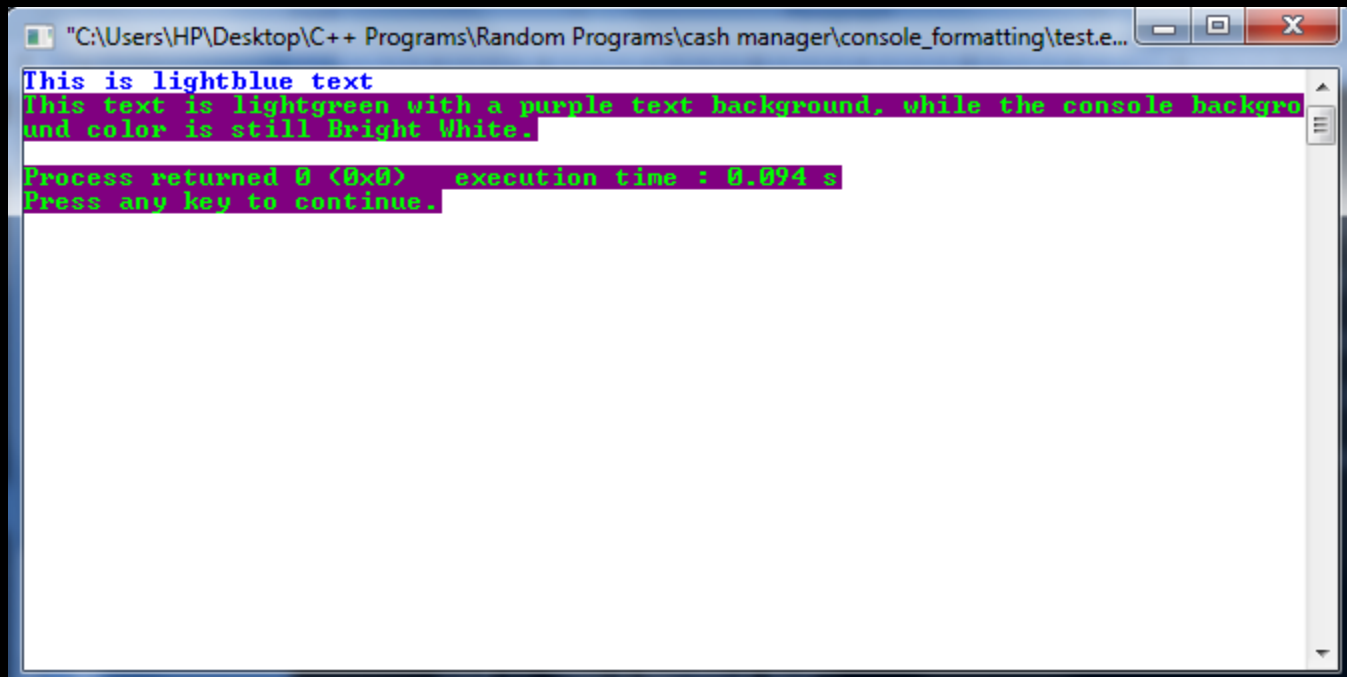
Similar to setBgClr(), this function sets the foreground color of the console. It takes a string argument as the name of the foreground color. The valid color names are:

"Black", "Blue", "Green", "Cyan", "Red", "Purple", "Yellow", "White", "Gray", "L Blue", "L Green", "L Cyan", "L Red", "L Purple", "L Yellow", "Bright White"

The function can also take a 2nd optional string argument as background color name of the 'text only'. This background color is only applied to the text only but not to the whole console. If the color name is not valid the text background is same as the background color of the console. The function returns a Boolean value true if a valid foreground-color name is used. If the foreground color name does not match with any of the given valid names the current foreground color is restored and the function returns a Boolean false value. Example:

```
setBgClr("Bright White"); //background color is set to bright white
setFgClr("L Blue"); //foreground color is set to lightblue
cout << "This is lightblue text";
setFgClr("L Green", "Purple"); //foreground color is now set to lightgreen and text color is set to purple
cout << endl << "This text is lightgreen with a purple text background, while the console background color is still Bright White." << endl;
```

OUTPUT:



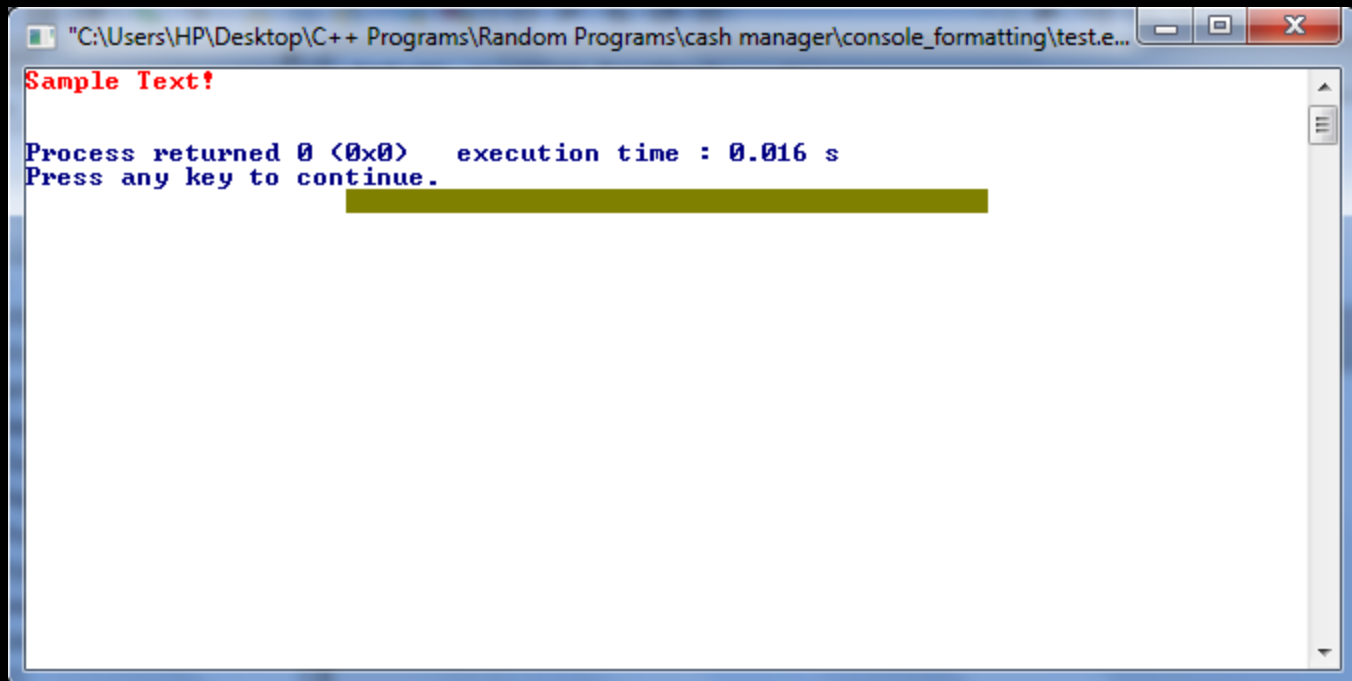
```
"C:\Users\HP\Desktop\C++ Programs\Random Programs\cash manager\console_formatting\test.e...
This is lightblue text
This text is lightgreen with a purple text background, while the console background color is still Bright White.
Process returned 0 (0x0)   execution time : 0.094 s
Press any key to continue.
```

7. colorArea() (only for windows)

This function colors the foreground and background (if specified) of a specified area anywhere in the console. It takes 4 compulsory and 1 optional arguments. The first argument is a string value representing the foreground color to be applied. The valid color names are those already specified in the setBgClr() and setFgClr() functions. The second argument is a short int type representing the x-coordinate of the console from where the coloring should be started. The third argument is again a short int type representing the y-coordinate of the console from where the coloring should be started. The third argument is an int type representing the number of units to be colored after the starting position/coordinates. The fourth optional coordinate takes a string value as the name of the background color of the area specified. Example:

```
cout << "Sample Text!" << endl << endl;
colorArea("L Red", 0, 0, 12); //foreground color of 12 units starting from 1st col and 1st row is set to light red.
colorArea("L Red", 20, 5, 40, "Yellow"); /*starting from 21st col and 6th row, 40 units' foreground-color is set to light red
and background-color is set to yellow. */
```

OUTPUT:



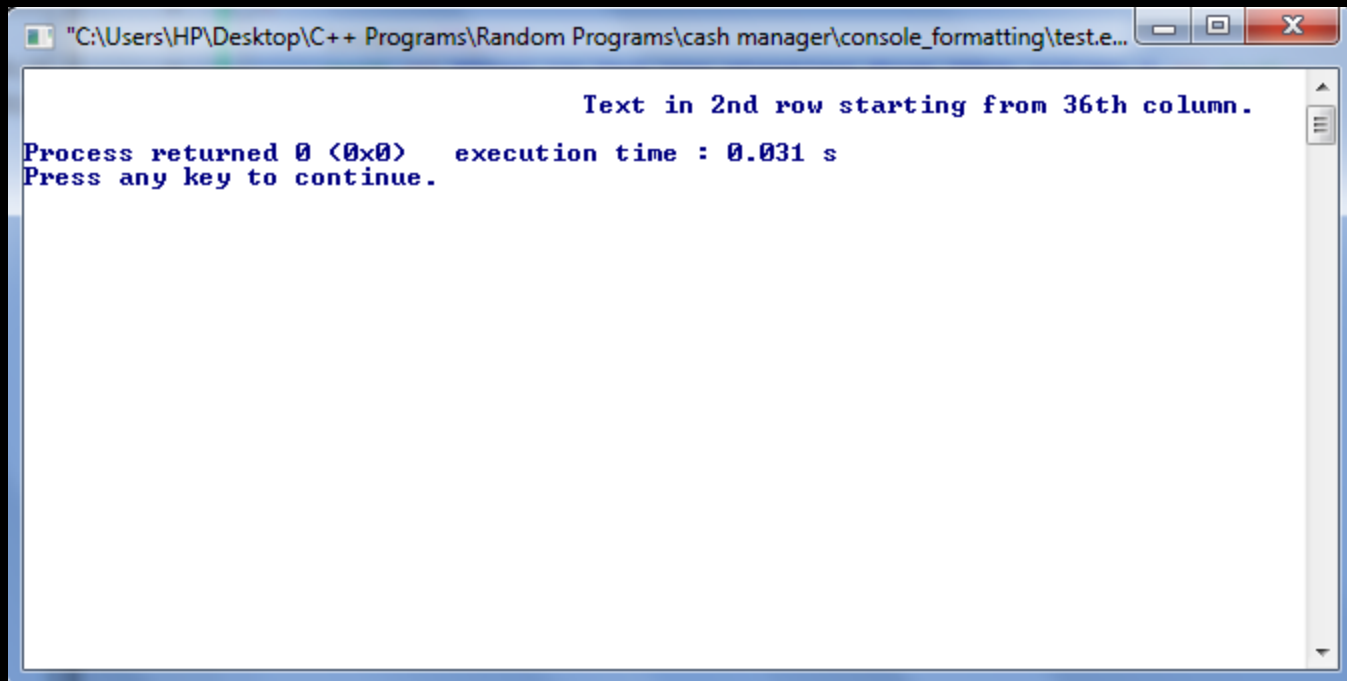
```
"C:\Users\HP\Desktop\C++ Programs\Random Programs\cash manager\console_formatting\test.e...
Sample Text!
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

8. Locate() (only for windows)

This function sets the position of cursor at a specified point within the console. It takes two short-int type arguments. The first one specifies x-coordinate or column number, while the second specifies y-coordinate or row number of the new cursor position. Remember row and column number starts from 0. Example:

```
Locate(35,1);  
cout << "Text in 2nd row starting from 36th column." << endl;
```

OUTPUT:



The screenshot shows a Windows console window titled "C:\Users\HP\Desktop\C++ Programs\Random Programs\cash manager\console_formatting\test.e...". The console output is as follows:

```
Text in 2nd row starting from 36th column.  
Process returned 0 (0x0)   execution time : 0.031 s  
Press any key to continue.
```

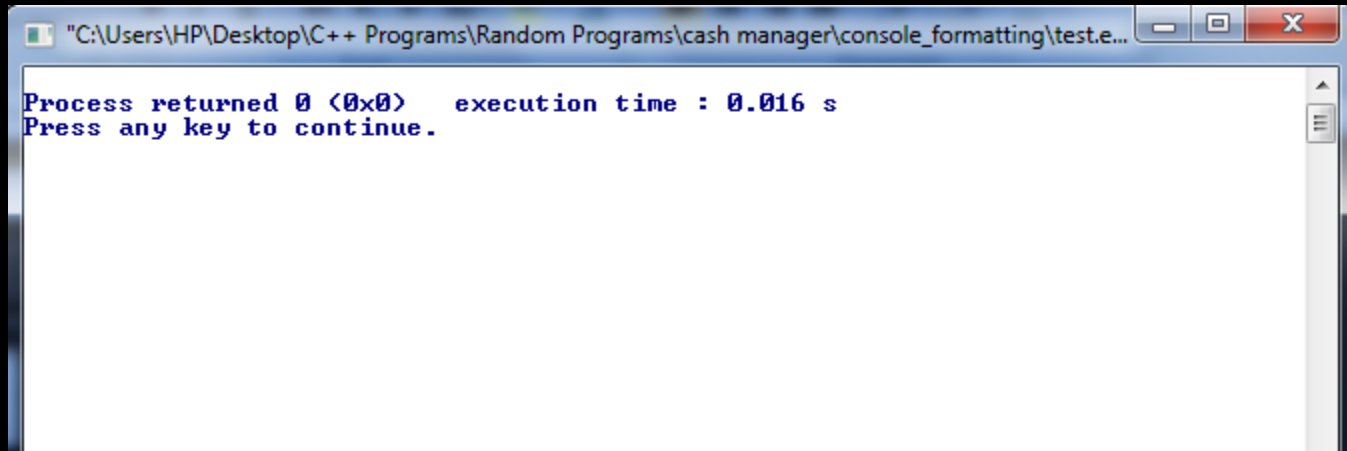
The text "Text in 2nd row starting from 36th column." is displayed starting at column 36 on the second row of the console, demonstrating the effect of the `Locate(35,1)` function call.

9. cursorVisAndSize() (only for windows)

This function shows/hide as well sets the size of the cursor. It takes first Boolean argument. If this is true the cursor is shown and if it is false the cursor is not shown on the console. The second short int type argument is optional and specifies the length of the cursor. A valid length is an integer from 1 to 100. If the second argument is not given, the default cursor size is retained. Example:

```
cursorVisAndSize(false); //hide the cursor from console
```

OUTPUT:



1. No cursor on the screen

```
cursorVisAndSize(true,100); //show the cursor with maximum size(the size of one character/unit)
```

(output can't be demonstrated through image in this case).

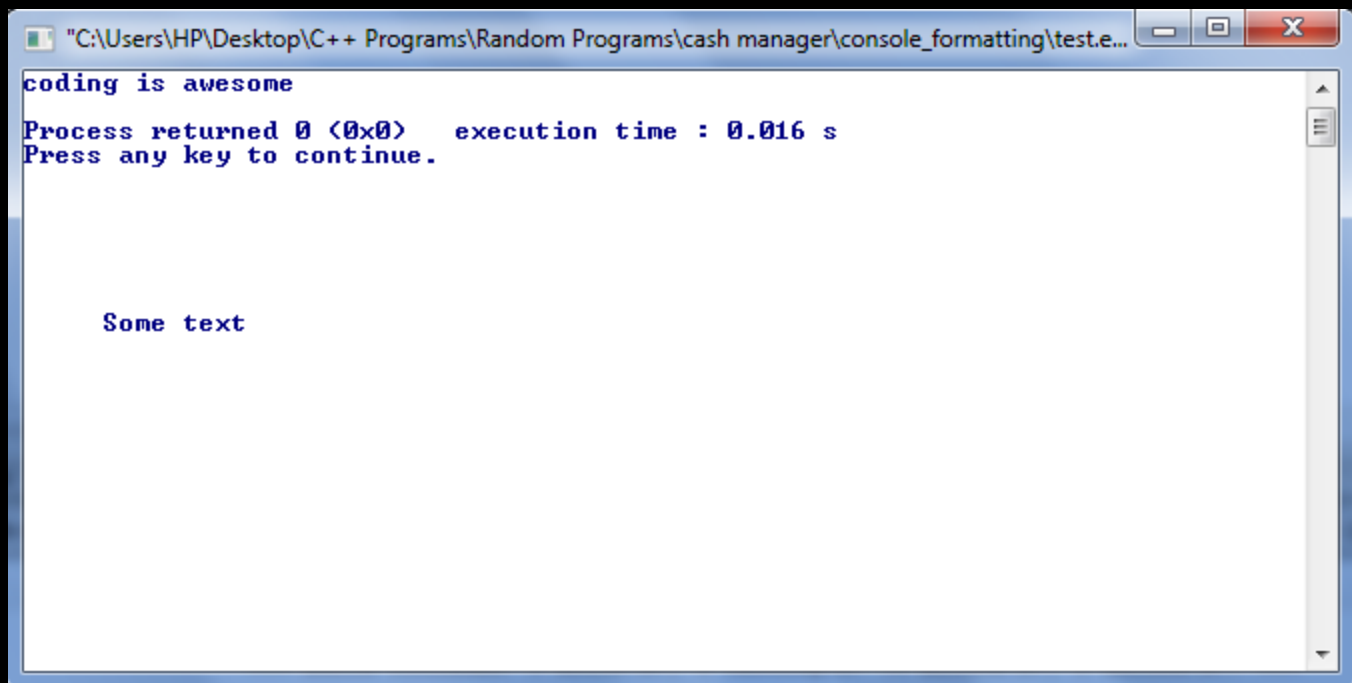
10. WriteTextAtLoc() (only for windows)

This function writes text at any location (specified by the x-y coordinates) within the console. The first argument accepts the string value which is to be written. The second argument accepts a short int value as the x-coordinate or column number of the console. The third and last argument accepts a short int value as the y-coordinate or row number of the console.

Unlike Locate() function which just moves the cursor position to the specific point the WriteTextAtLoc() function writes the text at the given location while retaining the cursor's original position. This comes into handy when you have to update some text on the screen so that the original cursor position remains the same. Example:

```
cout << "coding is ";  
WriteTextAtLoc("Some text",5,10); //wrote "Some text" at location 5,10 while cursor's position does not change  
cout << "awesome" << endl;
```

OUTPUT:



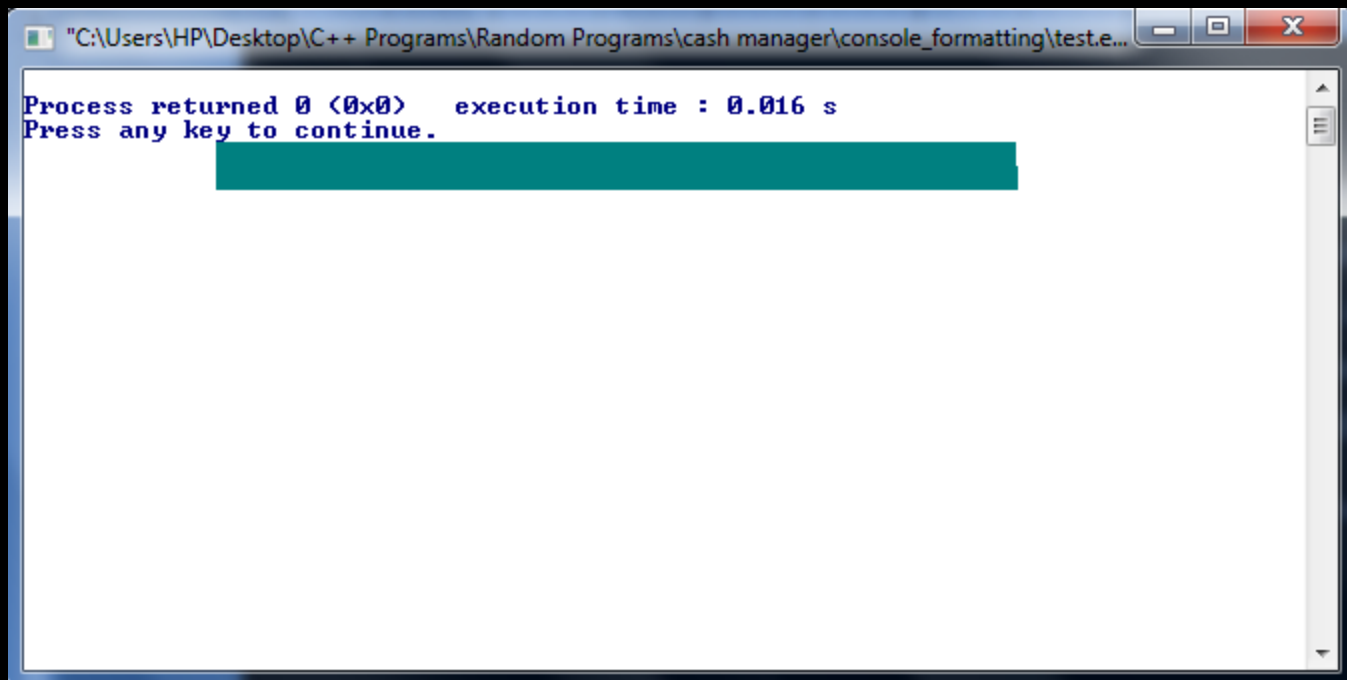
```
"C:\Users\HP\Desktop\C++ Programs\Random Programs\cash manager\console_formatting\test.e...  
coding is awesome  
Process returned 0 (0x0) execution time : 0.016 s  
Press any key to continue.  
  
Some text
```

11. clrLine() (only for windows)

To draw a portion of specified color on the console screen. Accepts 5 arguments. First is a string type referring to the color name of the portion. Valid color names have already been defined previously in the setBgClr() function. The second and third are short int type and refer to the x and y coordinates respectively of the starting point of the colored portion/line. The fourth and fifth are again short int type referring to the x and y coordinates respectively of the ending point of the portion. Example:

```
clrLine("Cyan",12,3,62,5); /*a Cyan colored portion 40 units wide and 2 units in height starting from the location (12,3) and ending at (62,5)*/
```

OUTPUT:



```
"C:\Users\HP\Desktop\C++ Programs\Random Programs\cash manager\console_formatting\test.e...  
Process returned 0 (0x0) execution time : 0.016 s  
Press any key to continue.  
[Cyan bar]
```

If you find something confusing or poorly described or any mistakes in this documentation feel free to tell me at maazproductions25@gmail.com

