

Homework #5 : printf() implementata con UART

Michele Corrias

23 aprile 2021

1 Consegnna

Lo studente implementi una funzione di libreria che consenta di rispondere affermativamente alle seguente domanda:

"Friends who have studied C should know the standard library functions printf() and scanf(), the former for printing information to the console, and the latter for reading characters from the keyboard to the program. If printf() is called in the program of the microcontroller, where will it be displayed? The answer is unknowable, because the microcontroller does not have a console, but we can use its peripherals to implement printf(), such as LCD or serial port (the serial port is then connected to the computer to display the print information). The serial port is basically the majority of microcontrollers, and the LCD is not necessarily, so we usually use the serial port to print content.

So as long as it is a serial port microcontroller, can you print the information by calling printf()?"

2 Premessa

Questo homework è stato risolto con successo su Windows 10 \x64, tramite l'utilizzo del converter *Silicon Labs CP2104*. Ho scaricato ed installato i driver ufficiali per Windows: https://www.silabs.com/documents/public/software/CP210x_Universal_Windows_Driver.zip. Ho inserito il dongle nella porta USB del computer: questo veniva riconosciuto sulla porta COM5 correttamente. Quindi, per permettere il corretto funzionamento, ho impostato sulla porta seriale i parametri che avrei utilizzato nel mio programma (figura 1). Successivamente ho collegato la board al dongle tramite 3 jumper:

- il TX pin UART del microcontrollore (nel mio caso PA2 perché USART2) con l'RXD pin del converter
- il RX pin UART del microcontrollore (nel mio caso PA3 perché USART2) con il TXD pin del converter
- il GND pin della board al ground del converter

Infine ho collegato la board al pc tramite il cavo USB Mini B (figura 2).

Per visualizzare i risultati del mio programma ho utilizzato l'eumulatore di terminale TeraTerm,

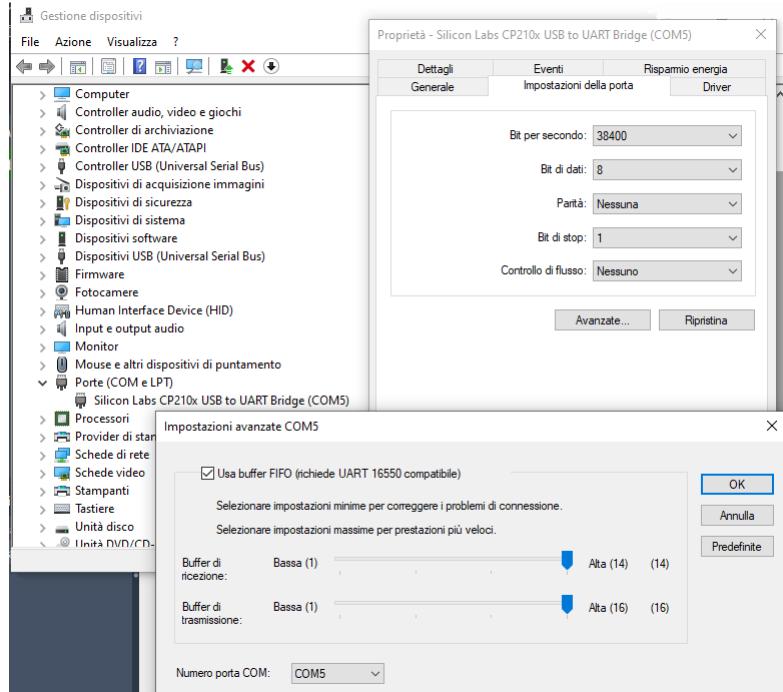


Figura 1: Porta COM 5

settato sulla porta seriale COM5 dove risiede il dongle, con un setup in linea con i parametri di configurazione utilizzati (figura 3).

3 Soluzione

La maggior parte dei microcontrollori di default non ha il concetto di console. Per usare le funzioni standard di libreria come la `printf` occorre eseguire un processo di *retargeting di IO*. Occorre quindi ridefinire le chiamate di sistema necessarie (`_write()` nel caso della `printf()`) per reindirizzare gli stream standard STDIN, STDOUT e STDERR sulla board.

Per ottenere questo risultato creiamo un nuovo STM32 project. Ci sono diverse periferiche UART sulla STM32F04 Discovery, ma noi scegliamo la USART2, associata all’interfaccia ST-LINK, che fornisce una porta COM virtuale tramite USB.

Quindi dall’IDE STM32CubeMX:

- il primo passo è abilitare la periferica USART2 all’interno della vista *Pinout*, selezionando la categoria *Connectivity* e scegliendo USART2
- poi a destra nel box *Mode* selezionare la voce *Asynchronous*
- entrambi i pin PA2 e PA3 verranno automaticamente evidenziati in verde
- quindi impostare nel box *Configuration* i parametri di configurazione

Si è scelto di mantenere una configurazione in linea con la porta COM, ovvero:

- *BaudRate* = 38400 Bps perché è la più alta velocità di trasmissione con errore correlato ancora nullo
- una *word length* di 8 bit (compreso il bit di parità)

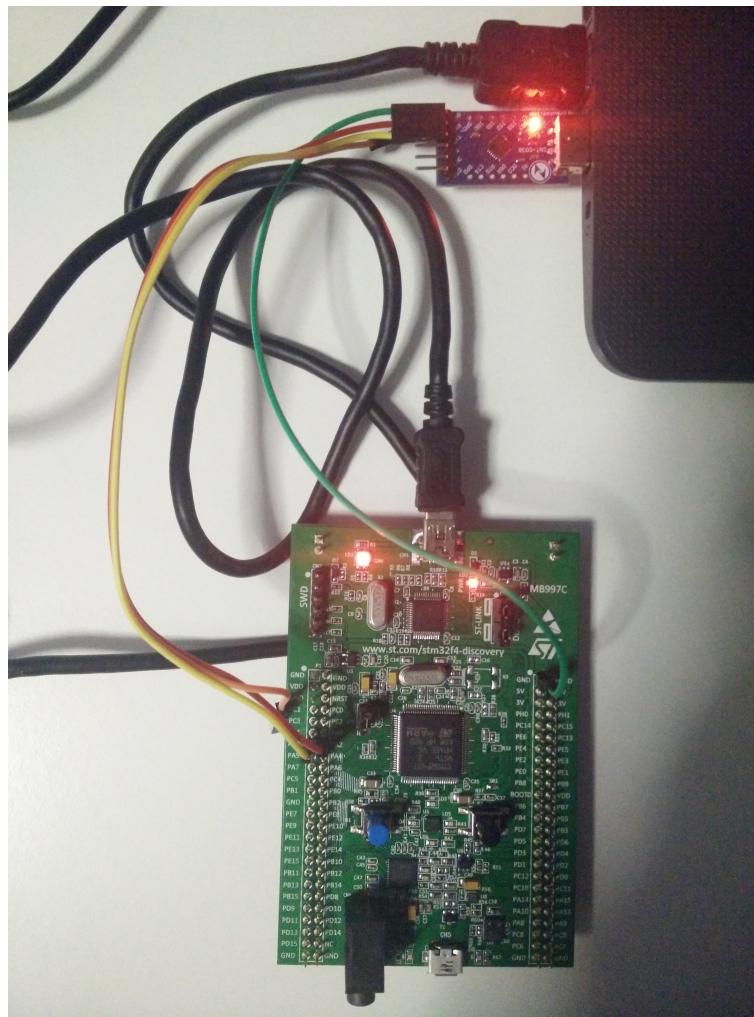


Figura 2: Collegamenti

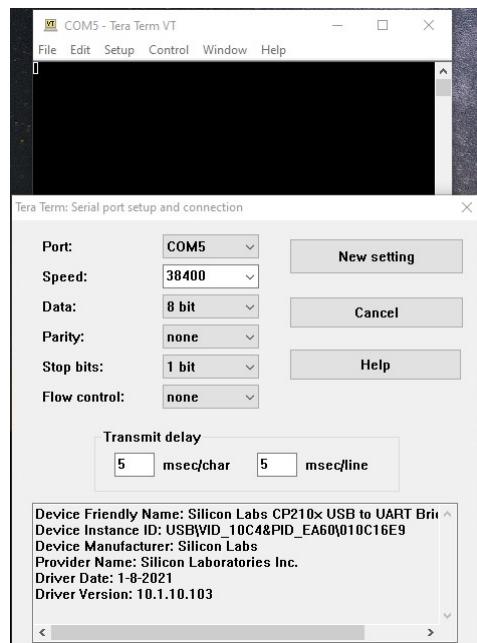


Figura 3: TeraTerm

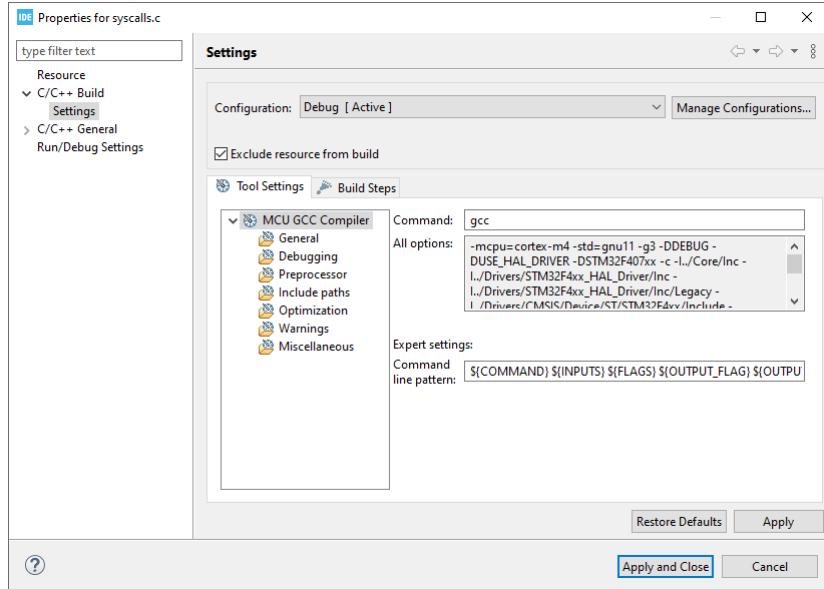


Figura 4: syscall.c

- nessun *parity bit*
- un solo *stop bit*
- nessun *Hardware Flow Control*
- *Oversampling* = 16

Ogni periferica progettata per scambiare dati con il mondo esterno deve essere correttamente associato ai GPIO corrispondenti; cioè dobbiamo configurare i pin USART2 TX e RX. Observando lo schema della Discovery possiamo riscontrare che i pin USART2 TX e RX sono rispettivamente PA2 e PA3.

3.1 header

La prima cosa che dobbiamo fare è disabilitare `syscalls.c`. Questo file infatti definisce molte delle stesse funzioni che stiamo cercando di ridefinire, e quindi non disabilitandolo, si ottengono degli errori di "definizione multipla" in fase di compilazione.

Per fare ciò facciamo click con il tasto destro del mouse sul file da disabilitare, selezioniamo *Properties*, quindi *C/C++ Build > Settings* e flaggare *Exclude resource from build* (figura 4). Quindi cliccare *Apply and Close* ed il nome del file dovrebbe apparire ora disabilitato. Questo consentirà di non includere il file durante la compilazione ed il linking.

Successivamente creiamo e salviamo un nuovo file `retarget.h` nella cartella `Inc`, con il seguente codice:

```

1  /*
2   *  retarget.h
3   *
4   */
5 #ifndef _RETARGET_H_
6 #define _RETARGET_H_
7

```

```

8 #include "stm32f4xx_hal.h"
9 #include <sys/stat.h>
10
11 void RetargetInit(UART_HandleTypeDef *huart);
12 int _isatty(int fd);
13 int _write(int fd, char* ptr, int len);
14 int _close(int fd);
15 int _lseek(int fd, int ptr, int dir);
16 int _read(int fd, char* ptr, int len);
17 int _fstat(int fd, struct stat* st);
18
19 #endif // #ifndef _RETARGET_H_

```

3.2 source

Ora creiamo e salviamo un nuovo file `retarget.c` nella cartella `Src`, che sostituisce il file disabilitato `syscall.c` con le `syscall` modificate per mezzo delle primitive `UART`:

```

1 /*
2  *  retarget.c
3  *
4  */
5 #include <ansi.h>
6 #include <syslist.h>
7 #include <errno.h>
8 #include <sys/time.h>
9 #include <sys/times.h>
10 #include <limits.h>
11 #include <signal.h>
12 #include <../Inc/retarget.h>
13 #include <stdint.h>
14 #include <stdio.h>
15
16 #if !defined(OS_USE_SEMIHOSTING)
17
18 #define STDIN_FILENO 0
19 #define STDOUT_FILENO 1
20 #define STDERR_FILENO 2
21
22 UART_HandleTypeDef *gHuart;
23
24 void RetargetInit(UART_HandleTypeDef *huart) {
25     gHuart = huart;
26
27     /* Disable I/O buffering for STDOUT stream, so that
28      * chars are sent out as soon as they are printed. */
29     setvbuf(stdout, NULL, _IONBF, 0);
30 }
31
32 int _isatty(int fd) {
33     if (fd >= STDIN_FILENO && fd <= STDERR_FILENO)
34         return 1;
35
36     errno = EBADF;
37     return 0;
38 }

```

```

40 int _write(int fd, char* ptr, int len) {
41     HAL_StatusTypeDef hstatus;
42
43     if (fd == STDOUT_FILENO || fd == STDERR_FILENO) {
44         hstatus = HAL_UART_Transmit(gHuart, (uint8_t *) ptr, len, HAL_MAX_DELAY);
45     }
46     if (hstatus == HAL_OK)
47         return len;
48     else
49         return EIO;
50 }
51 errno = EBADF;
52 return -1;
53
54 int _close(int fd) {
55     if (fd >= STDIN_FILENO && fd <= STDERR_FILENO)
56         return 0;
57
58     errno = EBADF;
59     return -1;
60 }
61
62 int _lseek(int fd, int ptr, int dir) {
63     (void) fd;
64     (void) ptr;
65     (void) dir;
66
67     errno = EBADF;
68     return -1;
69 }
70
71 int _read(int fd, char* ptr, int len) {
72     HAL_StatusTypeDef hstatus;
73
74     if (fd == STDIN_FILENO) {
75         hstatus = HAL_UART_Receive(gHuart, (uint8_t *) ptr, 1, HAL_MAX_DELAY);
76         if (hstatus == HAL_OK)
77             return 1;
78         else
79             return EIO;
80     }
81     errno = EBADF;
82     return -1;
83 }
84
85 int _fstat(int fd, struct stat* st) {
86     if (fd >= STDIN_FILENO && fd <= STDERR_FILENO) {
87         st->st_mode = S_IFCHR;
88         return 0;
89     }
90
91     errno = EBADF;
92     return 0;
93 }
94
95 #endif // #if !defined(OS_USE_SEMIHOSTING)
```

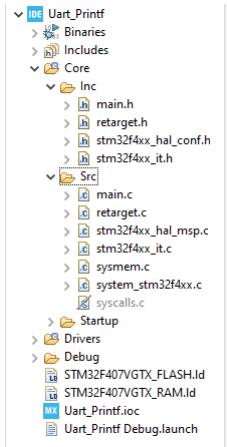


Figura 5: Struttura del progetto

La struttura delle directory del progetto dovrebbe essere simile alla figura 5.

3.3 main

Ora scriviamo e salviamo un file `main.c`, dove includiamo i file header `stdio.h` e `retarget.h`, e testiamo l'uso di `printf()`:

```

1  /* USER CODE BEGIN Header */
2  /**
3   *  main.c
4   */
5  /* USER CODE END Header */
6  /* Includes
7   * -----
8   */
9  /* Private includes
10  * -----
11 */
12 /* USER CODE BEGIN Includes */
13 #include <stdio.h>
14 #include "retarget.h"
15 /* USER CODE END Includes */
16
17 /* Private typedef
18  * -----
19 */
20 /* USER CODE BEGIN PTD */
21 /* USER CODE END PTD */
22
23 /* Private define
24  * -----
25 */
26 /* USER CODE BEGIN PD */
27 /* USER CODE END PD */
28
29 /* Private macro
30  * -----
31 */
32 /* USER CODE BEGIN PM */
33 /* USER CODE END PM */
34
35 /* Private variables
36  * -----
37 */

```

```

28 UART_HandleTypeDef huart2;
29
30 /* USER CODE BEGIN PV */
31 /* USER CODE END PV */
32
33 /* Private function prototypes
   ----- */
34 void SystemClock_Config(void);
35 static void MX_GPIO_Init(void);
36 static void MX_USART2_UART_Init(void);
37 /* USER CODE BEGIN PFP */
38
39 /* USER CODE END PFP */
40
41 /* Private user code
   ----- */
42 /* USER CODE BEGIN 0 */
43 /* USER CODE END 0 */
44
45 /**
 * @brief The application entry point.
 * @retval int
 */
46 int main(void)
{
47     /* USER CODE BEGIN 1 */
48     char buf[100];
49     /* USER CODE END 1 */
50
51     /* MCU Configuration
       ----- */
52
53     /* Reset of all peripherals, Initializes the Flash interface and the
       Systick. */
54     HAL_Init();
55
56     /* USER CODE BEGIN Init */
57     /* USER CODE END Init */
58
59     /* Configure the system clock */
60     SystemClock_Config();
61
62     /* USER CODE BEGIN SysInit */
63     /* USER CODE END SysInit */
64
65     /* Initialize all configured peripherals */
66     MX_GPIO_Init();
67     MX_USART2_UART_Init();
68     /* USER CODE BEGIN 2 */
69     RetargetInit(&huart2);
70     /* USER CODE END 2 */
71
72     /* Infinite loop */
73     /* USER CODE BEGIN WHILE */
74     while (1)
75     {
76         printf("\r\nInserisci il tuo nome e premi INVIO: ");
77         scanf("%s", buf);
78     }
79
80
81

```

```

82     printf( "\r\nBenvenuto , %s !\r\n" , buf );
83     /* USER CODE END WHILE */
84
85     /* USER CODE BEGIN 3 */
86 }
87 /* USER CODE END 3 */
88 }
89
90 /**
91 * @brief System Clock Configuration
92 * @retval None
93 */
94 void SystemClock_Config(void)
95 {
96     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
97     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
98
99     /** Configure the main internal regulator output voltage
100 */
101    __HAL_RCC_PWR_CLK_ENABLE();
102    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
103    /* Initializes the RCC Oscillators according to the specified parameters
104     * in the RCC_OscInitTypeDef structure.
105 */
106    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
107    RCC_OscInitStruct.HSISite = RCC_HSI_ON;
108    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
109    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
110    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
111    {
112        Error_Handler();
113    }
114    /* Initializes the CPU, AHB and APB buses clocks
115 */
116    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
117                                |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
118    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
119    RCC_ClkInitStruct.AHBClockDivider = RCC_SYSCLK_DIV1;
120    RCC_ClkInitStruct.APB1ClockDivider = RCC_HCLK_DIV1;
121    RCC_ClkInitStruct.APB2ClockDivider = RCC_HCLK_DIV1;
122
123    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
124    {
125        Error_Handler();
126    }
127 }
128 /**
129 * @brief USART2 Initialization Function
130 * @param None
131 * @retval None
132 */
133 static void MX_USART2_UART_Init(void)
134 {
135
136     /* USER CODE BEGIN USART2_Init_0 */
137     /* USER CODE END USART2_Init_0 */
138
139     /* USER CODE BEGIN USART2_Init_1 */

```

```

140  /* USER CODE END USART2_Init 1 */
141  huart2.Instance = USART2;
142  huart2.Init.BaudRate = 38400;
143  huart2.Init.WordLength = UART_WORDLENGTH_8B;
144  huart2.Init.StopBits = UART_STOPBITS_1;
145  huart2.Init.Parity = UART_PARITY_NONE;
146  huart2.Init.Mode = UART_MODE_TX_RX;
147  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
148  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
149  if (HAL_UART_Init(&huart2) != HAL_OK)
150  {
151      Error_Handler();
152  }
153  /* USER CODE BEGIN USART2_Init 2 */
154  /* USER CODE END USART2_Init 2 */
155 }

156
157 /**
158 * @brief GPIO Initialization Function
159 * @param None
160 * @retval None
161 */
162 static void MX_GPIO_Init(void)
163 {
164     GPIO_InitTypeDef GPIO_InitStruct = {0};
165
166     /* GPIO Ports Clock Enable */
167     __HAL_RCC_GPIOA_CLK_ENABLE();
168
169     /* Configure GPIO pin : PA0 */
170     GPIO_InitStruct.Pin = GPIO_PIN_0;
171     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
172     GPIO_InitStruct.Pull = GPIO_NOPULL;
173     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
174 }

175
176 /* USER CODE BEGIN 4 */
177 /* USER CODE END 4 */
178
179 /**
180 * @brief This function is executed in case of error occurrence.
181 * @retval None
182 */
183 void Error_Handler(void)
184 {
185     /* USER CODE BEGIN Error_Handler_Debug */
186     /* User can add his own implementation to report the HAL error return
187      state */
188     __disable_irq();
189     while (1)
190     {
191     }
192     /* USER CODE END Error_Handler_Debug */
193 }

194 #ifdef USE_FULL_ASSERT
195 /**
196 * @brief Reports the name of the source file and the source line number

```

```

197     *           where the assert_param error has occurred.
198     * @param file: pointer to the source file name
199     * @param line: assert_param error line source number
200     * @retval None
201     */
202 void assert_failed(uint8_t *file, uint32_t line)
203 {
204     /* USER CODE BEGIN 6 */
205     /* User can add his own implementation to report the file name and line
206      number,
207      ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
208           line) */
209     /* USER CODE END 6 */
210 }
211 #endif /* USE_FULL_ASSERT */

```

Ora buildiamo il progetto e lo lanciamo, avendo avuto cura prima di aprire *TeraTerm* ed averlo connesso alla porta seriale corretta come descritto in precedenza. Il programma stampa sulla console una richiesta di inserimento del nostro nome (non compare a video anche quello che si digita) e successivamente ci saluta, come mostrato nella figura 6.

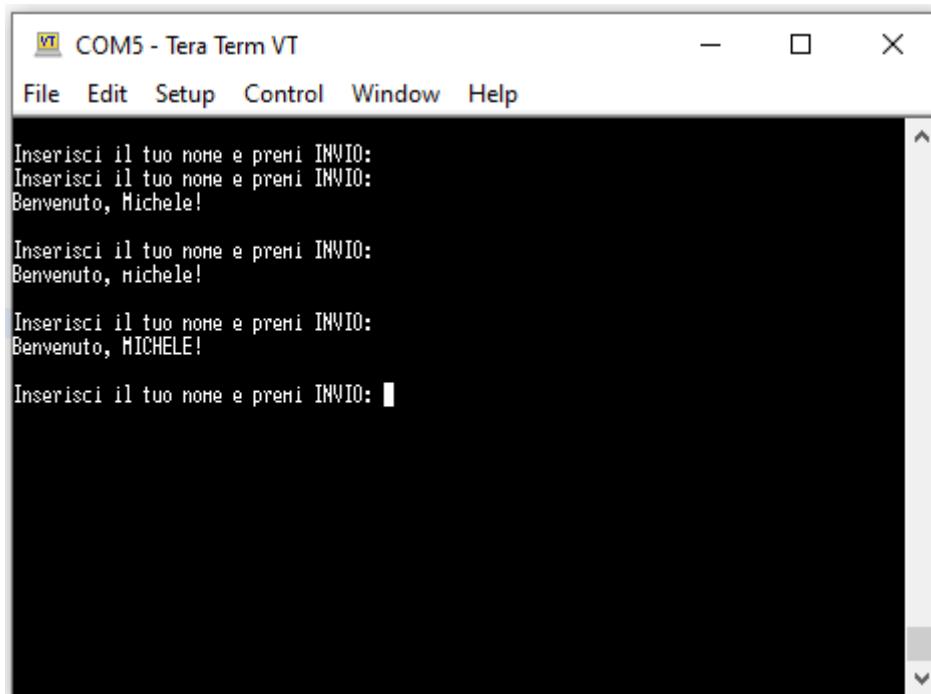


Figura 6: Output del programma