```rust
use std::env::Args;
use std::io;
use std::path::MAIN_SEPARATOR;
use std::time::{Duration, Instant};

fn main() {

    /* DICHIARAZIONE DI UNA VARIABILE IMMUTABILE i32 */
    let v: i32 = 123; //Non è possibile riassegnare il valore --> v = 5 è un ERRORE

    /* DICHIARAZIONE DI UNA VARIABILE MUTABILE i32 */
    let mut m: i32 = 123;
    m = 5;

    /* DICHIARAZIONE DI UNA VARIABILE IMMUTABILE f64 */
    let d: f64 = 1.3278;

    /* DICHIARAZIONE DI UNA VARIABILE IMMUTABILE f32 */
    let f: f32 = 1.3278f32;

    /* SEPARAZIONE DELLE CIFRE MEDIANTE '_' */
    let one_million = 1_000_000;

    println!("{}, {}, {}, {}, {}",v, m,d, f, one_million);

    /* DEFINIZIONE DI UNA TUPLA NON RIASSEGNABILE */
    let t: (i32, bool) = (123,false);
    println!("{:?}",t);

    /* ACCESSO AL PRIMO ELEMENTO DELLA TUPLA */
    let i = t.0;
    println!("{}", i);

    /* DEFINIZIONE DI UNA TUPLA RIASSEGNABILE */
    let mut u =  (3.14, 2.71);

```

```
37      /* MODIFICA DEL VALORE IN POSIZIONE 1 */
38      println!("{}",u.1);
39      u.1 = 0.0;
40      println!("{}",u.1);
41
42      /* RIFERIMENTO IN LETTUTURA*/
43      let mut i = 32;
44      let r = &i; //La variabile r prende in prestito
    il valore di i, che non potrà essere modificato
    finchè tale riferimento esiste --> i++ ERRORE
45      println!("{}",*r);
46
47      /*RIFERIMENTO IN LETTURA E SCRITTURA*/
48      let mut j = 32; //j deve essere necessariamente
    mutabile
49      let r1 = &mut i;
50      *r1 = *r1 + 1;
51      println!("{}", *r1); //Quando r1 prende in
    prestito il valore di j questo non può essere
    utilizzato se non da r1 (mutua esclusività) -->
    println!("{}",i} ERRORE
52
53      /* BOX<T> */
54      let mut b = Box ::new((5,2) ); //Definizione del
    Box
55      println!("{:?}",*b);
56
57      /* ARRAY */
58      let array: [i32; 5] = [1,2,3,4,5]; //Array
    contenente 5 elementi di tipo intero
59      let length = array.len(); //Metodo che
    restituisce la lunghezza di un array
60      let item = array[3]; //item <- elemento in
    posizione 3
61      println!("{:?} {} {}",array, length, item);
62
63      /* SLICE IMMUTABILE */
64      let s1: &[i32] = &array; //Copia dell'array nello
     slice s1
65      let s2 = &array[0..2]; //Copia dei valori nelle
    posizioni da 0 a 2 (escluso) nello slice s2
```

```rust
    let s3 = &array[2..]; //Copia dei valori dalla posizione 2 alla fine dell'array
    println!("{:?} {:?} {:?}", s1, s2, s3);

    /* SLICE MUTABILE */
    let mut arraym :[i32; 5] = [10,11,12,13,14];
    let s4: &[i32] = &mut arraym[0..2]; //Slice immutabile che contiene gli elementi dalla posizione 0 alla posizione 2
    println!("{:?}", s4);

    /*ARRAY ALLOCATO DINAMICAMENTE: Vec<T>*/
    let mut v: Vec<i32> = Vec::new();
    v.push(2);
    v.push(4);
    println!("{:?}", v);
    let slice = &mut v; //Creazione di uno slice del vettore v
    slice[1] = 8; //Modifica del secondo valore del vettore e dello slice (puntano allo stesso array)
    println!("{:?}", v);

    /* STRINGHE */
    let hello: &str = "hello,"; //Stringa memorizzata nella memoria statica
    println!("{:?}",hello);

    let mut string = String::new(); //Creazione di una stringa vuota(puntatore invalido, size = 0, capacity = 0)
    string.push_str(hello); //Stringa -> ptr: punta alla posizione dell'heap in cui viene allocata la memoria, size: 6, capacity: 8
    string.push_str(" world!"); //Stringa -> ptr: punta alla posizione dell'heap in cui viene allocata la memoria, size: 13, capacity: 16
    println!("{:?}",string);

    let str1 = String::from("some text"); //Creazione di una stringa inizializzata
    println!("{:?}",str1);
```

```
 94     let str2 = "some text".to_string(); //Creazione
    di una stringa inizializzata
 95     println!("{:?}",str2);
 96     let str3 = str2.to_string(); //Creazione di una
    stringa inizializzata con un'altra stringa
 97     println!("{:?}", str3);
 98
 99     /* METODI DELLE STRINGHE */
100     let object = str2.as_str(); //Ricavo di un
    oggetto di tipo &str2 da un oggetto String
101     println!("{}",object);
102
103     let mut stringm = String::from("Hello, ");
104     stringm.push_str("Raffaele"); //Inserimento in
    coda della stringa passata come parametro
105     stringm.insert_str(0,"Riccardo: "); //
    Inserimento nella posizione specificata della
    stringa passata come parametro
106     stringm.remove(16); //Eliminazione del carattere
     in posizione idx
107     println!("{}", stringm);
108     stringm.clear(); //Stringa svuotata
109     println!("{}", stringm);
110
111     let mut stringm1 = "hi raffaele".to_string();
112     let stringm2 = stringm1.to_uppercase(); //
    Conversione in maiuscolo
113     println!("{}", stringm2);
114
115     let stringm3 = stringm2.replace("RAFFAELE", "
    RICCARDO"); //Sostituzione di un blocco
116     println!("{}", stringm3);
117
118     let mut stringm4 = "         HI       ".
    to_string();
119     println!("{}", stringm4);
120     println!("After trim(), string length is {}",
    stringm4.trim().len());
121     println!("{:?}", stringm4.trim()); //
    Eliminazione di spaziature iniziali e finali
122
```

```
123     /* FUNZIONI */
124     fn add_numbers(x:i32, y:i32) -> i32 {
125         x + y //Non c'è il ";" finale!
126     }
127     let result = add_numbers(3,6);
128     println!("{}", result);
129 
130     /* ISTRUZIONI ED ESPRESSIONI:
131         - let, let mut --> Istruzioni
132         - Un blocco racchiudo tra {...} è un' espressione e restituisce il valore corrispondente
133           all'ultima espresssione a condizione che non sia terminata da ";"
134         - if ... else ... --> Espressione
135         - loop ... -> Espressione infinita caratterizzata da:
136                             - break returned_value ( opzionale) -> Consente di uscire dal loop
137                             - continue -> Consente di saltare un'istrzione in un'iterazione
138         - while ...
139         - for var in expression {code}
140     */
141     fn find_number(n:i32) -> i32 {
142         let mut count = 0;
143         let mut sum = 0;
144         loop {
145             count += 1;
146             if count % 5 == 0 {
147                 continue; //Ignora multipli di 5
148             }
149             if count % 3 == 0 {sum += 1} else {sum += 0}; //Conta i multipli di 3
150             if sum == n {
151                 break //Esci dal loop se il numero di multipli di tre è pari a n (parametro)
152             }
153         }
154         count
155     }
156     println!("Multipli: {}", find_number(5)); //
```

```
156 Invocazione della funzione
157 
158     /* CICLI INTERNI ED ESTERNI */
159     'outer: loop {
160         println!("Entrato nel ciclo esterno");
161         'inner: loop {
162             println!("Entrato nel ciclo inerno");
163             break //Esci dal ciclo interno ed esntra
     in quello esterno
164             println!("INTERNO");
165         }
166         println!("ESTERNO");
167         break
168     }
169     println!("Terminato il ciclo esterno");
170 
171     use std::time::{Duration, Instant}; //Importo
    della libreria standard +
172 
173     let mut counter = 0;
174     let time_limit = Duration::new(1,0); //Crea una
    durata di 1s
175     let start = Instant::now(); //Determina l'ora
    attuale
176     while (Instant::now() - start) < time_limit{
177         counter += 1;
178     }
179     println!("{}", counter);
180 
181     /* INTERVALLI
182         - a..b -> Intervallo semiaperto
183         - c..=d -> Intervallo chiuso
184         - .. -> Indica tutti i valori possibili per
    un dato dominio
185         - a.. -> Indica tutti i valori a partire da
    a (incluso)
186         - ..b -> Indica tutti i valori fino a b (
    escluso)
187         - ..=c -> Indica tutti i valori fino a c (
    incluso)
188         - d..e -> Indica tutti i valori tra d(
```

```
188 incluso) ed e(escluso)
189         - f..=g -> Indica tutti i valori tra f(
    incluso) ed g(incluso)
190     */
191
192     for n in 1..10 {
193         println!("{}",n); //Stampa i numeri da 1 a 9
194     }
195
196     let names = ["Bob", "Frank", "Ferris"];
197     for name in names.iter() {
198         println!("{}", name); //Stampa i tre nomi
199     }
200
201     for name in &names[..=1] {
202         println!("{}", name); //Stampa i primi due
    nomi
203     }
204
205     for (i,n) in names.iter().enumerate() {
206         println!("names[{}]:{}", i, n);
207     }
208
209     /* MATCH */
210     let item = 15;
211     let s = match item {
212         0 => "zero",
213         10 ..=20 => "Tra 10 e 20",
214         40 | 80 => "40 o 80",
215         _ => "Altro"
216     };
217     println!("{}",s);
218
219     let values = [1,2,3];
220     match &values[..] { //Crea uno slice con tutti
    gli elementi
221         //Contiene almeno un elemento e il primo
    elemento è 0
222         &[0, ..] => println!("Comincia con 0"),
223         //Contiene almeno un elemento, l'ultimo
    valore è compreso tra 3 e 5
```

```
224         &[.., v@ 3..=5] => println!("Finisce con {}"
    , v),
225         //Contiene almeno due elementi
226         &[_,v,..] => println!("Il secondo valore è
     {}", v),
227         //Contiene un solo elemento
228         &[v] => println!("Ha un solo elemento: {}",
    v),
229         //Non contiene elementi
230         &[] => println!("Slice vuoto")
231     }
232
233     /* POSSESSO */
234     let mut vp = Vec::with_capacity(4); //Creazione
    di un vettore di capacità 4 -> vp possiede il
    vettore
235     for i in 1..=5 {
236         vp.push(i);
237     }
238     println!("{:?}", vp); //Quando vp esce dal
    proprio scope sintattico, si occupa di rilasciare le
     risorse che possiede (array sull'heap + contenuto)
239
240     /* MOVIMENTO */
241     let mut strm1 = "hello".to_string();
242     println!("strm1: {}", strm1);
243     let strm2 = strm1; //Le risorse di strm1
    viengono spostate su strm2
244     println!("strm2: {}", strm2);
245     //strm1.to_uppercase() -> ERRORE: strm1 NON è
    più accessibile dopo aver ceduto le sue risorse
246
247     /* COPIA */
248     let mut num1 :i32 = 3;
249     let mut num2 :i32 = num1; //Copia del valore di
    num1 in num2
250     num2 +=1;
251     println!("num1: {}, num2: {}", num1, num2); //
    Non esiste alcun vincolo su num1
252
253     /* CLONAZIONE */
```

```
254     let mut sc1 = "hi".to_string();
255     let sc2 = sc1.clone();
256     sc1.push('!');
257     println!("sc1: {}, sc2: {}", sc1, sc2);
258 
259     /* RIFERIMENTI MUTABILI:
260         - A partire da una variabile che possiede un valore è possibile estrarre
261           UN SOLO riferimento mutabile per volta: let r = &mut v;
262         - E' possibile creare un riferimento mutabilie SOLO SE la variabile che possiede
263           il dato è mutabile a sua volta
264     */
265     let mut s = String::new();
266     { //Dal momento che s è statat dichiarata prima del blocco durerà di più rispetto a quanto presente nel blocco
267         let m = &mut s; //Blocco in cui s non può essere usato in nessun modo
268     } //m è movibile, è possibile ridurlo ad un riferimento semplice (non può modificare s), m non è copiabile
269 
270     let snm = String::from("hello");
271     {
272         let r = &snm; //r è copiabile, ma non può essere mutabile o movibile
273     }
274 
275     /* RIGA DI COMANDO */
276     use std::env::args; //Contiene parametri di tipo String
277 
278     let args: Vec<String> = args().skip(1).collect();
279     if args.len() > 0 { //args.len sostituisce argc e restituisce il numero di parametri
280         println!("Esistono parametri!");
281     }
282 
```

```
283     /* CLAP
284     use clap::Parser;
285 
286     #[derive(Parser, Debug)] /*Parse -> Struttura valorizzata analizzando quello che c'è nella command line e riempie i campi
287                                                 con i dettagli inseriti (name e count).
288                                     Debug -> Compilatore genera una versione stampabile*/
289     #[command(version, long_about = None)]
290 
291     struct Args {
292         #[arg(short, long)] //Decorazione -> Nella command line se si trova -n (o --name), quello che viene dopo si valorizza dentro name
293         name: String,
294         #[arg(short, long, default_value_t = 1)] // Decorazione -> Nella command line se si trova -c (o --count), quello che viene dopo si valorizza dentro count (se non si trova -c o --count mette 1)
295         count: u8,
296     }
297 
298     let args = Args::parse(); //Invoca il metodo statico parse che provvede a interrogare la command line e vede tutto quello che trova e restituisce un' istanza della struttura valorizzata
299     for _ in 0..args.count {
300         println!("Hello {}!", args.name);
301     }*/
302 
303     /* STDIO
304     use std::io;
305 
306     let mut strs = String::new();
307     if io::stdin().read_line(&mut strs).is_ok() {
308         //read_line: metodo che necesseita come parametro di un &mut di una stringa
309         //in cui andrà ad inserire ciò che legge e restituisce un result che può essere:
```

```
310         //  - ok -> Lettura andata a buonfine
311         //  - error -> Errore durante la lettura
312         println!("Got {}", strs.trim());
313     }else {
314         println!("Failed to read line!");
315     }
316 
317     //In alternativa è possibile usare unwrap che restituisce cosa ha letto se tutto è andato bene;
318     //altrimenti genera un "panik" se qualcosa è andato storto, interropendo il programma.
319     io::stdin().read_line(&mut strs).unwrap();
320     println!("Got {}", strs.trim());
321     //expect è un alternativa all'unwarp che consente di interrompere il programma, ma con una stringa scelta dal programmatore */
322 
323 }
324 
```