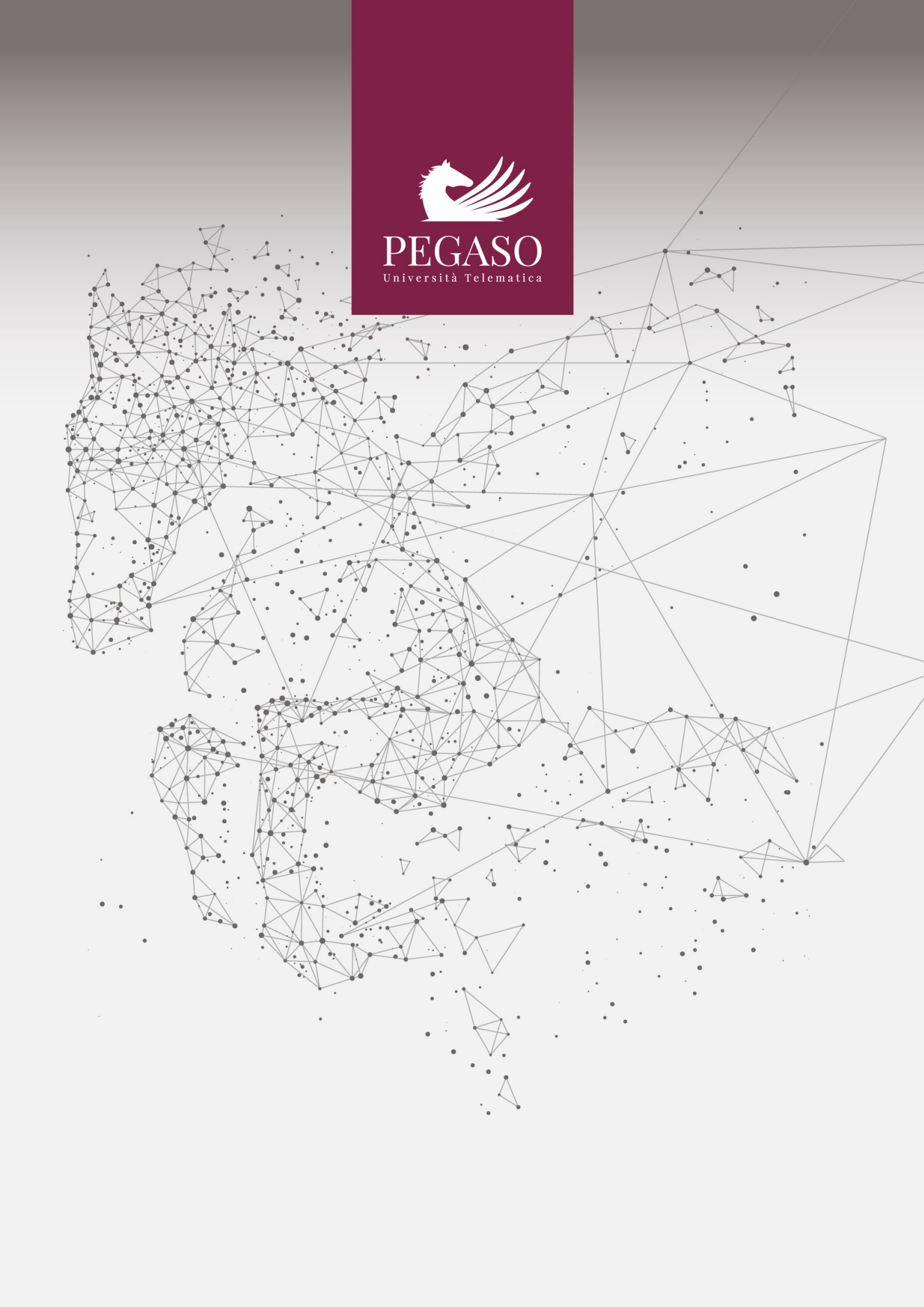
PEGASO
Università Telematica

Indice

1. File di intestazione

Ogni libreria standard ha un file di intestazione corrispondente, contenente i prototipi per tutte le funzioni in quella libreria e le definizioni dei vari tipi di dati e delle costanti necessarie a quelle funzioni.

La seguente tabella elenca in ordine alfabetico alcuni file di intestazione della Libreria Standard che si possono includere nei programmi.

Il C standard include ulteriori file di intestazione.

Intestazione	Illustrazione
`<assert.h>`	Contiene informazioni per aggiungere istruzioni di diagnostica che agevolano il debugging (ricerca di errori e correzione) dei programmi.
`<ctype.h>`	Contiene prototipi di funzione per le funzioni che verificano certe proprietà dei caratteri e per le funzioni che si possono usare per convertire lettere minuscole in lettere maiuscole e viceversa.
`<errno.h>`	Definisce le macro che sono utili per segnalare condizioni di errore.
`<float.h>`	Contiene i limiti per i valori in virgola mobile del sistema.
`<limits.h>`	Contiene i limiti per i valori interi del sistema.
`<locale.h>`	Contiene i prototipi di funzione e altre informazioni che consentono di modificare il programma per l'ambiente locale nel quale viene eseguito. La nozione di ambiente locale consente a un sistema di elaborazione di trattare convenzioni diverse per esprimere dati come date, ore, quantità monetarie e grandi numeri dovunque nel mondo.
`<math.h>`	Contiene i prototipi di funzione per le funzioni della libreria math.

`<setjmp.h>`	Contiene i prototipi di funzione per le funzioni che permettono di non rispettare la normale sequenza di chiamata e ritorno di funzioni.
`<signal.h>`	Contiene i prototipi di funzione e le macro per gestire varie situazioni che possono insorgere durante l'esecuzione di un programma.
`<stdarg.h>`	Definisce le macro per trattare liste di argomenti per una funzione, il cui numero e i cui tipi non sono noti a priori.
`<stddef.h>`	Contiene le definizioni di tipo comunemente usate dal C per effettuare calcoli.
`<stdio.h>`	Contiene i prototipi di funzione per le funzioni della Libreria Standard di input/output, nonché le informazioni usate da queste.
`<stdlib.h>`	Contiene i prototipi di funzione per convertire numeri in testo e testo in numeri, per allocare memoria, per trattare numeri casuali e per altre funzioni di utilità.
`<string.h>`	Contiene i prototipi di funzione per le funzioni di elaborazione di stringhe.
`<time.h>`	Contiene i prototipi di funzione e i tipi per manipolare il tempo e le date.

Esempio con `limits.h` (contiene i limiti per i valori interi del sistema)

```
#include <stdio.h>
#include <limits.h>
int main(void)
{
   printf("Number of bits in a byte %d\n", CHAR_BIT);

   printf("Min value of SIGNED CHAR = %d\n", SCHAR_MIN);
```

```
    printf("Max value of SIGNED CHAR = %d\n", SCHAR_MAX);
    printf("Max value of UNSIGNED CHAR = %d\n", UCHAR_MAX);

    printf("Min value of SHORT INT = %d\n", SHRT_MIN);
    printf("Max value of SHORT INT = %d\n", SHRT_MAX);

    printf("Min value of INT = %d\n", INT_MIN);
    printf("Max value of INT = %d\n", INT_MAX);

    printf("Min value of CHAR = %d\n", CHAR_MIN);
    printf("Max value of CHAR = %d\n", CHAR_MAX);

    printf("Min value of LONG = %ld\n", LONG_MIN);
    printf("Max value of LONG = %ld\n", LONG_MAX);
}
```

Output:

```
Number of bits in a byte 8
Min value of SIGNED CHAR = -128
Max value of SIGNED CHAR = 127
Max value of UNSIGNED CHAR = 255
Min value of SHORT INT = -32768
Max value of SHORT INT = 32767
Min value of INT = -2147483648
Max value of INT = 2147483647
Min value of CHAR = -128
Max value of CHAR = 127
Min value of LONG = -9223372036854775808
```

Max value of LONG = 9223372036854775807

Potete creare file di intestazione personalizzati.

I file di intestazione definiti dal programmatore devono utilizzare anche l'estensione del nome di file `.h`.

Un file di intestazione definito dal programmatore può essere incluso usando la direttiva per il preprocessore `#include`.

Ad esempio, se il prototipo per la nostra funzione `square` fosse contenuto nel file di intestazione `square.h`, includeremmo questo file di intestazione nel nostro programma, usando la seguente direttiva in cima al programma stesso:

```
#include "square.h"
```

Notare che le intestazioni definite dal programmatore sono racchiuse tra virgolette ("") anziché tra parentesi angolari (<>).

Passare gli argomenti per valore e per riferimento

In molti linguaggi di programmazione ci sono due modi per passare gli argomenti: il passaggio per valore e il passaggio per riferimento.

Quando gli argomenti sono passati per valore, viene fatta una copia del valore dell'argomento, che viene passata alla funzione chiamata.

Le modifiche alla copia non incidono sul valore della variabile originaria nella funzione chiamante.

Quando un argomento è passato per riferimento, la funzione chiamante permette alla funzione chiamata di modificare il valore della variabile originaria.

Il passaggio per valore va usato ogni volta che la funzione chiamata non ha necessità di modificare il valore della variabile originaria della funzione chiamante.

Questo evita effetti secondari accidentali (modificazioni delle variabili) che sono di grande impedimento allo sviluppo di sistemi software corretti e affidabili.

Il passaggio per riferimento va usato solo con funzioni chiamate fidate che necessitano effettivamente di modificare le variabili originarie.

Nel C tutti gli argomenti sono passati per valore.

Vedremo che sarà possibile ottenere il passaggio per riferimento utilizzando operatori opportuni

Concentriamoci per ora sul passaggio per valore, sfruttando `return` come in questo esempio `square`:

```
int square(int x)
{
   // x e' una variabile locale
   return x * x;
}
```

Classi di memoria

Abbiamo usato identificatori per i nomi delle variabili.

Attributi delle variabili sono:

- nome, tipo, dimensioni e valore.

Usiamo ora gli identificatori anche come nomi per le funzioni definite dall'utente.

In realtà, ogni identificatore in un programma ha altri attributi, quali la classe di memoria, la permanenza in memoria, il campo di azione e il collegamento.

Il C fornisce gli specificatori della classe di memoria:

- `auto`
- `extern`
- `static`

- `_Thread_local` (aggiunto in C11, non lo considereremo)

La classe di memoria di un identificatore determina la sua permanenza in memoria, il suo campo di azione e il suo collegamento.

La permanenza in memoria di un identificatore è il periodo durante il quale l'identificatore esiste nella memoria.

Alcuni esistono per breve tempo, alcuni sono ripetutamente creati e distrutti e altri esistono per l'intera esecuzione del programma.

Il campo di azione (in inglese "scope") di un identificatore è dove l'identificatore può essere menzionato in un programma.

Alcuni si possono menzionare per tutto il programma, altri solo da porzioni di esso.

Il collegamento di un identificatore determina, per un programma con diversi file sorgente, se l'identificatore è conosciuto soltanto nel file sorgente corrente o in qualunque file sorgente con le opportune dichiarazioni.

Iniziamo ad esaminare le classi di memoria e la permanenza in memoria.

Gli specificatori della classe di memoria si suddividono tra permanenza in memoria automatica e permanenza in memoria statica.

La parola chiave auto è usata per dichiarare le variabili con permanenza in memoria automatica.

Le variabili con permanenza in memoria automatica sono create quando il controllo del programma entra nel blocco nel quale sono definite.

Esse esistono finché il blocco è attivo e sono distrutte quando il controllo del programma esce dal blocco.

Variabili locali

Solo le variabili possono avere permanenza in memoria automatica.

Le variabili locali di una funzione (quelle dichiarate nella lista dei parametri o nel corpo della funzione) hanno normalmente permanenza in memoria automatica.

La parola chiave auto dichiara esplicitamente le variabili con permanenza in memoria automatica.

Per impostazione predefinita, le variabili locali hanno permanenza in memoria automatica, così la parola chiave auto si usa raramente.

Per il resto del testo chiameremo le variabili con permanenza in memoria automatica semplicemente variabili automatiche.

Prestazioni

La tecnica di memorizzazione automatica è un mezzo per risparmiare memoria, poiché le variabili automatiche esistono solo quando ce n'è necessità.

Esse sono create quando si entra in una funzione e distrutte quando si esce da essa.

Classe di memoria statica

Le parole chiave extern e static si usano nelle dichiarazioni degli identificatori per le variabili e le funzioni con permanenza in memoria statica.

Gli identificatori con permanenza in memoria statica esistono dal momento in cui il programma inizia l'esecuzione fino a che il programma termina.

Per le variabili static la memoria è allocata e inizializzata soltanto una volta, prima che il programma inizi l'esecuzione.

Per le funzioni, il nome della funzione esiste quando il programma inizia l'esecuzione. Tuttavia, anche se le variabili e i nomi di funzione esistono dall'avvio dell'esecuzione del programma, ciò non significa che a questi identificatori si possa accedere per tutta la durata del programma.

La permanenza in memoria e il campo d'azione (dove si può usare un nome) sono aspetti separati.

Vi sono diversi tipi di identificatori con permanenza in memoria statica: gli identificatori esterni (come le variabili globali e i nomi di funzione) e le variabili locali dichiarate con lo specificatore della classe di memoria static.

Le variabili globali e i nomi di funzione sono della classe di memoria extern per impostazione predefinita.

Le variabili globali sono create ponendo le dichiarazioni di variabile all'esterno della definizione di qualsiasi funzione; esse mantengono i loro valori per tutta l'esecuzione del programma.

Le variabili globali e le funzioni possono essere menzionate da una qualunque funzione definita in seguito alle loro dichiarazioni o definizioni nello stesso file.

Questo giustifica l'uso di prototipi di funzione. Quando includiamo stdio.h in un programma che chiama printf, il prototipo di funzione è posto all'inizio del nostro file, per rendere noto il nome printf al resto del file.

Osservazione di ingegneria del software

Definire una variabile globale invece che locale permette che si verifichino effetti secondari non voluti quando una funzione che non ha necessità di accedere alla variabile la modifica accidentalmente o intenzionalmente.

In generale, le variabili globali vanno evitate, tranne in certe situazioni con critici requisiti di prestazioni.

Osservazione di ingegneria del software

Le variabili usate solo in una data funzione devono essere definite come variabili locali in quella funzione invece che come variabili esterne.

Le variabili locali dichiarate con la parola chiave static sono ancora conosciute solo nella funzione in cui sono definite, ma, diversamente dalle variabili automatiche, le variabili locali static mantengono il loro valore quando si esce dalla funzione.

La volta successiva che la funzione è chiamata, una variabile locale static contiene il valore che aveva prima che si uscisse dalla funzione la volta precedente.

L'istruzione seguente dichiara la variabile locale count come static e la inizializza a 1.

```
static int count = 1;
```

Se non vengono inizializzate esplicitamente, tutte le variabili numeriche con permanenza in memoria statica sono inizializzate automaticamente a zero.

Le parole chiave extern e static hanno un significato speciale quando sono esplicitamente

applicate a identificatori esterni. Nel Capitolo 14 esamineremo l'uso esplicito di extern e static con identificatori esterni e con programmi con più file sorgente.

Ottenere un valore intero pseudocasuale

Considerate la seguente istruzione:

```
i = rand();
```

La funzione rand genera un numero intero tra 0 e `RAND_MAX` (una costante simbolica definita nel file di intestazione <stdlib.h>).

Il C standard stabilisce che il valore di `RAND_MAX` deve essere almeno pari a 32767, che è il valore massimo per un intero di due byte (cioè 16-bit).

Su gcc di GNU, `RAND_MAX` vale 2147483647.

Se rand produce veramente interi a caso, ogni numero tra 0 e RAND_MAX ha un'uguale probabilità di essere scelto ogni volta che rand viene chiamata.

L'intervallo di valori prodotto direttamente da rand è spesso diverso da quello richiesto in una specifica applicazione.

Ad esempio, un programma che simula il lancio di una moneta può richiedere soltanto 0 per "testa" e 1 per "croce".

Un programma per il lancio di dadi che simula un dado di sei facce richiede numeri interi casuali da 1 a 6.

Lanciare un dado di sei facce

Per illustrare rand, sviluppiamo un programma che simula 20 lanci di un dado di sei facce e stampa il valore ottenuto per ogni lancio.

```
01  // Programma C
02  //  Interi casuali con spostamento e variazione di scala.
03  #include <stdio.h>
04  #include <stdlib.h>
05
06  int main(void)
07  {
08     // ripeti 20 volte
09     for (unsigned int i = 1; i <= 20; ++i) {
10
11        // scegli un numero casuale da 1 a 6 e stampalo
12        printf("%10d", 1 + (rand() % 6));
13
14        //  se contatore divisibile per 5, inizia una nuova riga
15        if (i % 5 == 0) {
16           puts("");
17        }
18     }
19  }
```

```
6  6  5  5  6
5  1  1  5  3
6  6  2  4  2
6  2  3  4  1
```

Il prototipo di funzione per la funzione rand si trova in `<stdlib.h>`.

Usiamo l'operatore di resto (`%`) congiuntamente a rand come segue

```
rand() % 6
```

per generare interi nell'intervallo da 0 a 5.

Questa operazione è chiamata variazione di scala.

Il numero 6 si dice fattore di scala.

Quindi effettuiamo uno spostamento dell'intervallo dei numeri generati aggiungendo 1 al nostro precedente risultato.

L'output conferma che i risultati stanno nell'intervallo da 1 a 6 (gli effettivi valori casuali scelti possono variare in base al compilatore).

Lancio di un dado a sei facce 60.000.000 di volte

Per verificare che questi numeri ricorrono approssimativamente con uguale probabilità, simuliamo 60.000.000 di lanci di un dado con il programma seguente.

Ogni intero da 1 a 6 deve comparire approssimativamente 10.000.000 di volte.

```
1  // Programma
2  // Lancio di un dado a sei facce 60.000.000 di volte.
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main(void)
7  {
```

```
 8     unsigned int frequency1 = 0; // contatore per il valore 1
 9     unsigned int frequency2 = 0; // contatore per il valore 2
10     unsigned int frequency3 = 0; // contatore per il valore 3
11     unsigned int frequency4 = 0; // contatore per il valore 4
12     unsigned int frequency5 = 0; // contatore per il valore 5
13     unsigned int frequency6 = 0; // contatore per il valore 6
14
15     // ripeti 60.000.000 di volte e riepiloga i risultati
16     for (unsigned int roll = 1; roll <= 60000000; ++roll) {
17        int face = 1 + rand() % 6; // numero casuale da 1 a 6
18
19           //   determina il valore di face e incrementa il
contatore
20        switch (face) {
21
22           case 1: // valore 1
23              ++frequency1;
24              break;
25
26           case 2: // valore 2
27              ++frequency2;
28              break;
29
30           case 3: // valore 3
31              ++frequency3;
32              break;
33
34           case 4: // valore 4
35              ++frequency4;
36              break;
37
38           case 5: // valore 5
```

```
39                ++frequency5;
40                break;
41
42             case 6: // valore 6
43                ++frequency6;
44                break; // opzionale
45          }
46       }
47
48       // stampa i risultati in formato tabellare
49       printf("%s%13s\n", "Face", "Frequency");
50       printf("   1%13u\n", frequency1);
51       printf("   2%13u\n", frequency2);
52       printf("   3%13u\n", frequency3);
53       printf("   4%13u\n", frequency4);
54       printf("   5%13u\n", frequency5);
55       printf("   6%13u\n", frequency6);
56    }
```

Face	Frequency
1	9999294
2	10002929
3	9995360
4	10000409
5	10005206
6	9996802

Come mostra l'output del programma, con una variazione di scala e uno spostamento dell'intervallo abbiamo usato la funzione rand per simulare realisticamente il lancio di un dado a sei facce.

Si noti l'uso dello specificatore di conversione `%s` per stampare le stringhe di caratteri "Face" e "Frequency" come intestazioni delle colonne (riga 49).

Dopo aver studiato gli array, mostreremo come sostituire elegantemente questa istruzione switch di 26 righe con un'istruzione di una singola riga.

Esercizio:

Ripetete più volte l'esecuzione del programma

Sono davvero numeri casuali?

Vedremo successivamente come la libreria time.h ci consentirà di generare numeri 'realmente' pseudocasuali

Riferimenti bibliografici

- Paul Deitel, Harvey Deitel, "Il linguaggio C – Fondamenti e tecniche di programmazione", Libro edito da Pearson Italia. Include anche utili esercizi di autovalutazione.