1. Applicazione dell'overloading dell'operatore <<

L'**operator overloading** (sovraccarico dell'operatore) indica la possibilità di definire nuove funzionalità ai vari operatori del linguaggio (++, – –, ecc.), in modo che il compilatore sia in grado di comprendere il loro impiego nei nuovi tipi di dati creati dal programmatore.
L'*operator overloading* non è altro che un *function overloading* applicato a funzioni particolari chiamate **operator** seguite dall'operatore a cui si riferiscono. Si può ottenere, per esempio, l'*operator overloading* per l'incremento unitario ++ oppure del segno + per la somma attraverso i *function overloading* delle funzioni *operator++* o *operator+* rispettivamente.

PROGETTO

Calcolo degli incassi settimanali di un negozio.

Per rappresentare i giorni della settimana utilizziamo un'enumerazione, assegnando i valori da 1 a 7 ai nomi simbolici dei giorni:

```
enum Giorni {LUN=1, MAR, MER, GIO, VEN, SAB, DOM};
```

Volendo poi gestire l'acquisizione dei dati tramite una ripetizione *for*, occorre ridefinire l'operatore ++ per l'incremento del contatore da LUN a SAB (giorni di apertura del negozio), in modo che possa essere applicato al nuovo tipo *Giorni*.
L'*overloading* dell'operatore ++ è realizzato con la funzione *operator++*, come mostrato nel programma seguente.

PROGRAMMA C++

```
// Giorni.cpp: incassi settimanali
#include <iostream>
using namespace std;

enum Giorni {LUN=1, MAR, MER, GIO, VEN, SAB, DOM};

Giorni operator++(Giorni& g, int i);

int main()
{
   float totale=0;
   float incasso;

   for(Giorni i=LUN;i<=SAB;i++) {
      cout << "Incasso del giorno " << i << ": ";
      cin >> incasso;
      totale += incasso;
   }
   cout << "Incasso totale = " << totale << endl;

   return 0;
}
```


```
Giorni operator++(Giorni& g, int i)
{
   i = g;
   g = static_cast<Giorni> (++i);

   return g;
}
```

Si noti che, nella funzione *operator++*, l'operatore di incremento unitario è utilizzato come *prefisso*, perché il valore di *i* deve essere calcolato prima del suo utilizzo.

È possibile applicare l'*overloading* agli operatori >> e << per le operazioni di input/output. Per esempio, nel programma precedente si potrebbe ridefinire l'operatore << in modo da visualizzare il valore simbolico del giorno, anziché il numero intero che compete ad esso nell'enumerazione. Questo tipo di *overloading* viene risolto con il progetto seguente che utilizza gli *stream*.

PROGETTO

Calcolo degli incassi settimanali.

Il programma utilizza un'enumerazione per rappresentare i giorni della settimana, assegnando i valori da 1 a 7 ai nomi simbolici dei giorni:

```
enum Giorni {LUN=1, MAR, MER, GIO, VEN, SAB, DOM};
```

Il programma dell'unità di apprendimento 4 viene ora ampliato con l'aggiunta dell'operatore << in modo da poter visualizzare i nomi dei giorni, durante l'acquisizione dei dati, anziché un numero intero da 1 a 7.
L'overloading dell'*operatore* << è codificato con una funzione nella quale l'operatore di output ha due parametri, rappresentati dal flusso di output che è un oggetto di classe **ostream** e dalla stringa da visualizzare.

```
ostream& operator<< (ostream& os, Giorni g)
{
   string s;

   switch(g) {
   case LUN:
      s = "LUN";
      break;
   case MAR:
      s = "MAR";
      break;
   case MER:
      s = "MER";
      break;
   case GIO:
      s = "GIO";
      break;
   case VEN:
      s = "VEN";
      break;
```


```
    case SAB:
        s = "SAB";
        break;
    case DOM:
        s = "DOM";
        break;
    default:
        s = "valore non corretto";
        break;
    }

    os << s;

    return os;
}
```

Si osservi che il tipo restituito dalla funzione *operator* << è una referenza allo stream *os* di tipo *ostream* per consentire anche operazioni di output ripetute, come la seguente:

```
cout << x << y << z;
```

Il programma completo è riportato di seguito.

PROGRAMMA C++

```
#include <iostream>
#include <string>
using namespace std;

enum Giorni {LUN=1, MAR, MER, GIO, VEN, SAB, DOM};
float incasso;

Giorni operator++ (Giorni&, int);
ostream& operator<< (ostream&, Giorni);

int main()
{
   float totale=0;

   for(Giorni i=LUN;i<=SAB;i++) {
       cout << "Incasso del giorno " << i << ": ";
       cin >> incasso;
       totale += incasso;
   }
   cout << "Incasso totale = " << totale << endl;

   return 0;
}

Giorni operator++ (Giorni& g, int i)
{

   i = g;
   g = static_cast<Giorni> (++i);

   return g;
}
```

```
ostream& operator<< (ostream& os, Giorni g)
{
   string s;

   switch(g) {
   case LUN:
      s = "LUN";
      break;
   case MAR:
      s = "MAR";
      break;
   case MER:
      s = "MER";
      break;
   case GIO:
      s = "GIO";
      break;
   case VEN:
      s = "VEN";
      break;
   case SAB:
      s = "SAB";
      break;
   case DOM:
      s = "DOM";
      break;
   default:
      s = "valore non corretto";
      break;
   }

   os << s;

   return os;
}
```

ESERCIZIO

Dopo aver definito un'enumerazione per i nomi dei primi sei mesi dell'anno, scrivere il codice per l'overloading dell'operatore << in modo da poter visualizzare i nomi dei mesi, anziché il loro indice all'interno dell'enumerazione.