# Esercizio 1 – Tipo enumerato

- Scrivere un programma in cui si dichiari un tipo enumerato **giorno**, rappresentante i giorni della settimana, e si stampino i valori dei 7 enumeratori (senza usare cicli). Quindi si definisca una variabile di tipo **giorno**, le si assegni uno degli enumeratori a piacere, e la si stampi.

# Esercizio 1 – Tipo enumerato

- **SOLUZIONE**

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

enum giorno_t     {lunedi, martedi, mercoledi, giovedi,
venerdi, sabato, domenica} ;

main()
{
      cout<<lunedi<<endl ;
      cout<<mercoledi<<endl ;
      cout<<domenica<<endl ;
      giorno_t giorno ;
      giorno = giovedi ;
      cout<<"giorno: "<<giorno<<endl ;
}
```

# Esercizio 2 – Tipo enumerato

- Scrivere un programma contenente una funzione **stampa_turno**, che abbia come parametro di ingresso un oggetto di tipo uguale al tipo di dato rappresentante i giorni della settimana. Dato il giorno della settimana passato in ingresso, la funzione stampa i turni di quel giorno.

- Supporre che siano stati fissati i seguenti turni:
- Lunedi', Mercoledi' e Venerdi': mattina e pomeriggio
- Martedi' e Giovedi': solo mattina
- Sabato e Domenica: riposo.

- Nel main non si legge nulla da stdin, ma la funzione deve essere invocata per stampare i turni dei giorni di Lunedi', Giovedi', Domenica.

# Esercizio 2 – Tipo enumerato

- **SOLUZIONE**

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

enum giorno_t {lunedi, martedi, mercoledi, giovedi, venerdi, sabato, domenica} ;

void stampa_turno(giorno_t g) {
        switch(g) {
        case lunedi:
        case mercoledi:
        case venerdi:
                cout<<"Mattina e pomeriggio"<<endl ; break ;
        case martedi:
        case giovedi:
                cout<<"Solo mattina"<<endl ; break ;
        case sabato:
        case domenica:
                cout<<"Riposo"<<endl ;    }
}
main(){
        cout<<"Lunedi': " ;       stampa_turno(lunedi) ;
        cout<<"Giovedi': " ;      stampa_turno(giovedi) ;
        cout<<"Domenica: " ;      stampa_turno(domenica) ;}
```

# Esercizio 3 – Tipo reale – Conversioni

- Scrivere un programma che legga due numeri INTERI, e stampi il risultato della divisione REALE tra i due numeri.

- Assumere che non si possa avere una espressione di tipo intero come rvalue in un assegnamento in cui a sinistra (lvalue) si ha una variabile di tipo reale.

Esempio:
Inserisci due valori interi: 5 2
5 / 2 = 2.5

# Esercizio 3 – Tipo reale – Conversioni

- **SOLUZIONE**

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

main(){
        int a, b ;
        cout<<"Inserisci due valori interi: " ;
        cin>>a>>b ;
        cout<<a<<" / "<<b<<" = "<<static_cast<double>(a)/b<<endl ;
}
```

**NOTA**: Faccio la conversione esplicita solo per a.
La divisione tra un double (*static_cast<double>(a)/*) e un intero (*b*) comporterà implicitamente il cast di *b* a tipo double.

# Esercizio 4 – Tipo reale – Conversioni

- Scrivere un programma che legga due numeri INTERI, e stampi il risultato della divisione REALE tra i due numeri.

- **Non utilizzare la  conversione esplicita per realizzare il programma.**

Esempio:
Inserisci due valori interi: 5 2
5 / 2 = 2.5

# Esercizio 4 – Tipo reale – Conversioni

- **SOLUZIONE**

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

main(){
        int a, b ;
        cout<<"Inserisci due valori interi: " ;
        cin>>a>>b ;

        double c = b;
        cout<<a<<" / "<<b<<" = "<<a/c<<endl ;
}
```

**NOTA**: La divisione tra un intero (*a*) e un double (*c*) comporterà implicitamente il cast di *a* a tipo double.

# Stampa numeri reali

- Nel caso di numeri con un alto numero di cifre dopo la virgola, o addirittura numeri decimali periodici
  - Viene stampato un alto numero di cifre dopo la virgola?
- Probabilmente no
- Si può controllare tale aspetto della formato di stampa mediante la funzione descritta nella prossima slide

# Manipolatore setprecision

**`setprecision(`**<*numero_cifre*>**`)`**

Setta il massimo numero di cifre per un numero in virgola mobile

- Bisogna includere **`<iomanip>`**
- L'effettivo output dipende dal formato (generale, scientifico, fisso)
- L'effetto è *persistente*: influenza tutte le prossime operazioni di uscita, fino alla prossima eventuale chiamata di *setprecision*

# Esercizio 5 – Tipo reale – Conversioni

- Utilizzando il manipolatore **setprecision**, modificare la soluzione del precedente esercizio per stampare molte più cifre dopo la virgola

# Esercizio 5 – Tipo reale – Conversioni

- **SOLUZIONE**

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

main(){
        int a, b ;
        cout << "Inserisci due valori interi: " ;
        cin >> a >> b ;

        double c = b;
        cout << a << " / “ << b << " = “ << setprecision(9) << a/c << endl ;
}
```

# Esercizio 6 – Tipo reale
## Perdita di informazione (troncamento)

- Scrivere un programma che legga in ingresso un numero reale e lo memorizzi in una variabile reale, quindi assegni il valore di tale variabile ad una variabile di tipo int, infine stampi il valore di entrambe le variabili.

- Assumere che in un assegnamento in cui si ha come lvalue (ossia a sinistra dell'assegnamento) l'indirizzo di una variabile di tipo int, NON SI POSSA avere come rvalue (ossia a destra dell'assegnamento) un valore di tipo reale.

# Esercizio 6 – Tipo reale
## Perdita di informazione (troncamento)

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

main()
{
      double a ;

      cout<<"Inserisci un valore reale: " ;     cin>>a ;

      // converto prima di assegnare
      int b = static_cast<int>(a) ;
      cout<<"reale == "<<a<<", intero == "<<b<<endl ;
}
```

# Esercizio 7 – Tipo reale
## Perdita di informazione (precisione)

- Scrivere un programma che legga un numero intero e lo memorizzi in una variabile di tipo int, poi assegni il valore di tale variabile ad una variabile di tipo float (non double), infine assegni il valore della seconda variabile ad una terza varibile di tipo int. Infine stampi il valore di tutte e tre le variabili.

- Usare conversioni esplicite per evitare warning. Scritto il programma, provare ad inserire ad esempio il numero 214748364. Viene segnalato qualche errore a tempo di esecuzione?

# Esercizio 7 – Tipo reale
## Perdita di informazione (precisione)

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

int main()
{
      int a ;
      cout << "Inserisci un valore intero: " ;
      cin >> a ;
      float b = static_cast<float>(a) ;
      int c = static_cast<int>(b) ;
      cout << "prima intera == “ << a <<
              ", reale == “ << b <<
              ", seconda intera == “ << c << endl ;
      return 0 ;
}
```

# Esercizio 7 – Tipo reale
## Perdita di informazione (precisione)

- La precisione della mantissa per il tipo float è inferiore rispetto alla precisione del tipo int. Dunque col tipo float non si riesce a rappresentare correttamente il numero 214748364.0
- NON viene segnalato alcun errore a tempo di esecuzione
- Si riesce a rappresentare lo stesso numero col tipo double?
- Perché?

# Esercizio 8 – Tipo reale
## Perdita di informazione (precisione)

- http://www.h-schmidt.net/FloatConverter/IEEE754.html
  - Provare a settare e resettare il bit del segno
  - Cercare la configurazione di bit che fa sì che l'esponente sia 0
  - Trovare la configurazione di bit che rappresenta il numero 1.5
  - Trovare anche la rappresentazione di 0.1
    - Tale numero è rappresentato in modo esatto?
    - Torneremo su questo argomento

# Esercizio 9 – Tipi enumerato e reale

Scrivere un programma per la gestione di un "ascensore impazzito" per un parco giochi.

L'ascensore si trova inizialmente in una posizione "0", a 20 metri dal suolo. Se ci si sposta verso l'altro la posizione aumenta di valore, mentre se ci si sposta verso il basso la posizione diminuisce di valore. Quindi, per esempio, la posizione 1 è a 21 metri d'altezza, mentre la posizione -2 è a 18 metri d'altezza.

Nell'attrazione l'ascensore sale o cade velocemente verso una nuova posizione, quindi frena bruscamente. L'ascensore è appeso a funi spostate da un motore comandato mediante impulsi elettrici: un impulso di voltaggio positivo o negativo fa ruotare il motore di un angolo di base in una direzione o nell'altra. A ciascuna rotazione di base, corrisponde uno spostamento di base dell'ascensore di 0.1 m verso l'alto o verso il basso.

# Esercizio 9 – Tipi enumerato e reale

Supporre per dare un impulso al motore e quindi far spostare l'ascensore occorra invocare una funzione ***singolo_impulso***, alla quale si passa la posizione attuale e la direzione (su o giu) verso cui effettuare uno spostamento di base.

Nel nostro programma supponiamo che la funzione si limiti a ritornare il valore della posizione passata come parametro, più o meno 0.1 a seconda della direzione passata anch'essa come parametro. NON UTILIZZARE UN INTERO PER LA DIREZIONE. E neanche un booleano.

Il main gestisce l'ascensore leggendo da stdin un numero intero rappresentante il numero di spostamenti di base da effettuare, ed invocando per un numero di volte oppportuno e per la direzione opportuna la funzione ***singolo_impulso***.  Numeri in ingresso negativi implicano spostamenti verso il basso, mentre numeri in ingresso positivi implicano spostamenti verso l'alto.

# Esercizio 9 – Tipi enumerato e reale

Nel nostro sistema di riferimento, le posizioni possibili vanno da -20.0 a +20.0 (ossia da 0 a 40 metri di altezza nel mondo reale). Se si richiedono un numero di spostamenti eccessivi il programma li limita comunque per restare nell'intervallo di posizioni [-20.0, +20.0] ed evitare che l'ascensore si schianti al suolo o sul motore stesso. Scrivere il programma ed assicurarsi che funzioni.

Testare con cura il programma.

# Esercizio 9 – Tipi enumerato e reale

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

enum direzione {su, giu};

float singolo_impulso (float pos, direzione d) {
          if (direzione == su)
                    return pos + 0.1;
          else
                    return pos - 0.1;
}

int main() {
          float posizione = 0.0;
          int n, limite; // numero di spostamenti e posizione finale
          direzione dir;

          do {
                    cout<<"\nInserire il numero di spostamenti: ";
                    cin>>n;

                    dir = (n>0) ? su : giu;
                    limite = static_cast<int>(posizione * 10) + n;
                    n = abs(n);

                    if ((limite <= -200) || (limite >= 200))
                              cout<<"\nIMPOSSIBILE MUOVERSI VERSO "<<dir<<" di "<<n<<" posizioni.";
                    else
                              for (int i = 0; i < n; i++)
                                        posizione = singolo_impulso (posizione, dir);

                    cout<<"\nLA NUOVA POSIZIONE E' "<<posizione<<endl;
          } while (n != 0);

          return 1;
}
```

# Esercizio 9 – Tipi enumerato e reale

**VARIANTE**:

Per rendere l'attrazione più emozionante, si decide di staccare automaticamente le funi quando l'ascensore si ferma, recuperando l'ascensore stesso con un gancio che scorre su un binario parallelo. Questo può succedere in qualsiasi posizione finale, tranne quella a distanza 1 dalla posizione 0, perché per motivi tecnici in tale posizione non si può effettuare l'aggancio.

Modificare il main per gestire il distacco dell'ascensore. In pratica per le posizioni sicure stampare "staccare le funi", mentre per quella pericolosa stampare "non staccare le funi!!!".

SOLUZIONE NON FORNITA