```
#ifndef _RANGE_H_
#define _RANGE_H_

#include <cassert>


template<typename T>
class Range {

 T start=0;
 T stop;
 T step=1;

 class Iterator {

  T index;
  T step;
  T stop;

 public:

  Iterator(T index, T step, T stop):index{index},step{step},stop{stop} {}

  T &operator*() { return index; }

  Iterator &operator++() {
   // to not exceed the stop value
   if ((index < stop && index + step <= stop) or (index > stop && index + step >= stop)) {
    index += step; } else { index = stop;
   } return *this;
  }

  auto operator<=>(const Iterator& other) const { return this->index <=> other.index; }
  bool operator!=(const Iterator& other) const { return this->index != other.index; }
 };

public:

 Range(T stop):stop{stop} {}

 Range(T start, T stop):start{start},stop{stop} {
  assert(start < stop && "error : the begin value should be lesser than the end value");
 }

 Range(T start, T stop, T step):start{start},stop{stop},step{step} {
  if ( start < stop ) {
   assert(step > 0 && "error : the step value should be positive");
  } else if ( start > stop ) {
   assert( step < 0 && "error : the step value should be negative");
  } else {
   assert (start == stop && "error : the start can't be equal to the end");
  }
  assert(step != 0 && "error : the step can't be 0");
 }

 Iterator begin() const { return Iterator(start, step, stop); }
 Iterator end() const { return Iterator(stop, step, stop); }

};

#endif
```