

Create an enumerated type for the days of the week.

Inductive day : Type :=

| monday : day

| tuesday : day

| wednesday : day

| thursday : day

| friday : day

| saturday : day

| sunday : day.

Write a function `next_weekday`.

```
Definition next_weekday (d:day) : day :=  
  match d with  
  | monday => tuesday  
  | tuesday => wednesday  
  | wednesday => thursday  
  | thursday => friday  
  | friday => monday  
  | saturday => monday  
  | sunday => monday  
end.
```

Prove that tuesday is two weekdays after
saturday.

Example test_next_weekday:

```
(next_weekday (next_weekday saturday)) = tuesday.
```

Create an enumerated type for booleans.

```
Inductive bool : Type :=  
  | true  : bool  
  | false : bool.
```


Define a boolean negation function.

```
Definition negb (b:bool) : bool :=  
  match b with  
  | true => false  
  | false => true  
end.
```

Define a function for boolean conjunction.

```
Definition andb (b1:bool) (b2:bool) : bool :=  
  match b1 with  
  | true => b2  
  | false => false  
end.
```

Define a function for boolean disjunction.

```
Definition orb (b1:bool) (b2:bool) : bool :=  
  match b1 with  
  | true => true  
  | false => b2  
end.
```

Define a polymorphic expression that can inhabit any type.

Definition admit {T: Type} : T. Admitted.

Define a function for boolean nand.

Definition nandb (b1:bool) (b2:bool) : bool := negb (andb b1 b2).

Create a type for natural numbers.

```
Inductive nat : Type :=  
  | O : nat  
  | S : nat -> nat.
```

Define a function for integer predecessor.

```
Definition pred (n:nat) : nat :=  
  match n with  
  | 0      => 0  
  | S n'   => n'  
  end.
```

Define a "minus two" function.

```
Definition minustwo (n:nat) : nat :=  
  match n with  
  | 0           => 0  
  | S 0         => 0  
  | S (S n')   => n'  
end.
```


Define a function for testing whether a natural number is even.

```
Fixpoint evenb (n:nat) : bool :=  
  match n with  
  | 0          => true  
  | S 0        => false  
  | S (S n')  => evenb n'  
end.
```

Define a function for testing whether a natural number is odd.

Definition oddb (n:nat) : bool := negb (evenb n).

Define a function for adding natural numbers.

```
Fixpoint plus (n:nat) (m:nat) : nat :=  
  match n with  
  | 0      => m  
  | S n'  => S (plus n' m)  
end.
```

Define a function for multiplying natural numbers.

```
Fixpoint mult (n m:nat) : nat :=  
  match n with  
    | 0      => 0  
    | S n'  => plus m (mult n' m)  
  end.
```


Define a function for subtracting natural numbers.

```
Fixpoint minus (n m:nat) : nat :=  
  match n, m with  
  | 0 , _      => 0  
  | S _ , 0    => n  
  | S n' , S m' => minus n' m'  
end.
```

Define a function for exponentiating natural numbers.

```
Fixpoint exp (base power:nat) : nat :=  
  match power with  
  | 0    => S 0  
  | S p => mult base (exp base p)  
end.
```