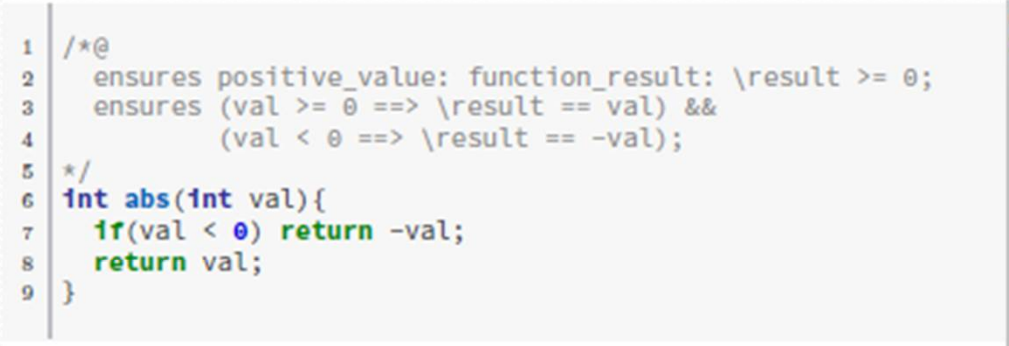
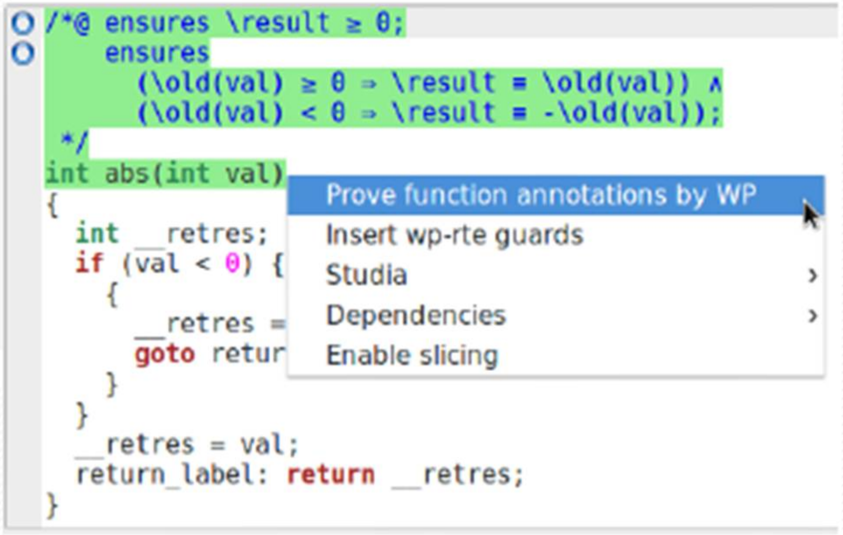
FRAMA-C: INTRODUCTION

Dr.S.Padmavathi
CSE, Amrita School of Engineering
Coimbatore

Frama-C GUI

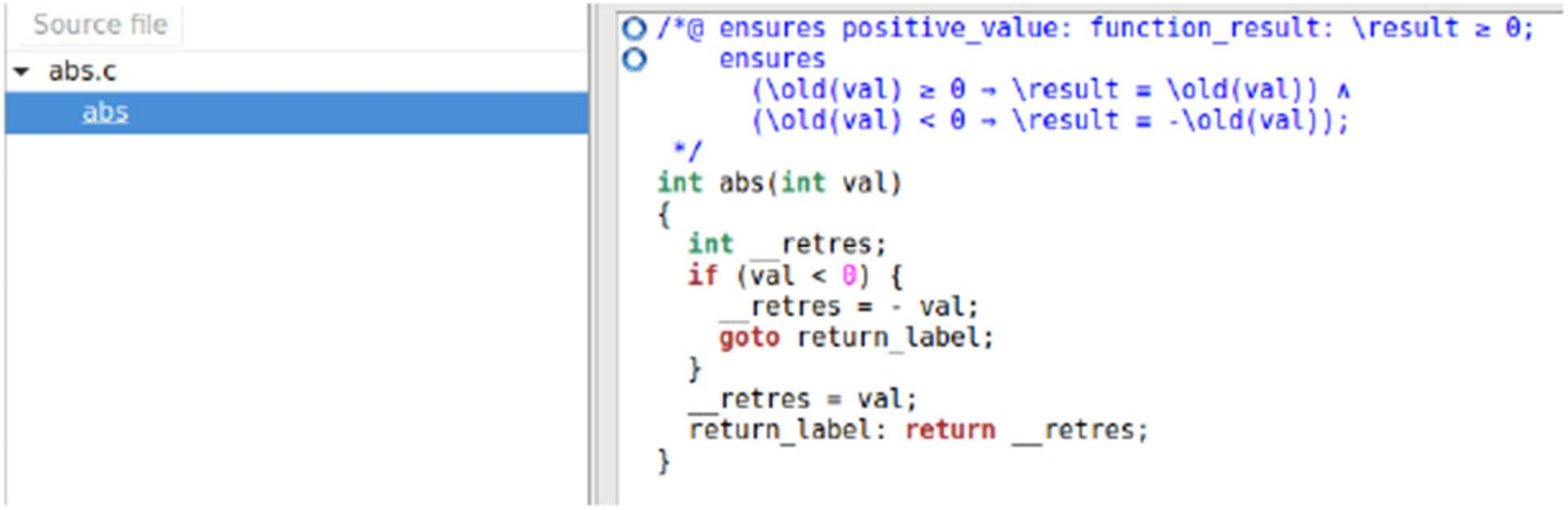


```
/*@
  ensures positive_value: function_result: \result >= 0;
  ensures (val >= 0 ==> \result == val) &&
          (val < 0 ==> \result == -val);
*/
int abs(int val){
  if(val < 0) return -val;
  return val;
}
```

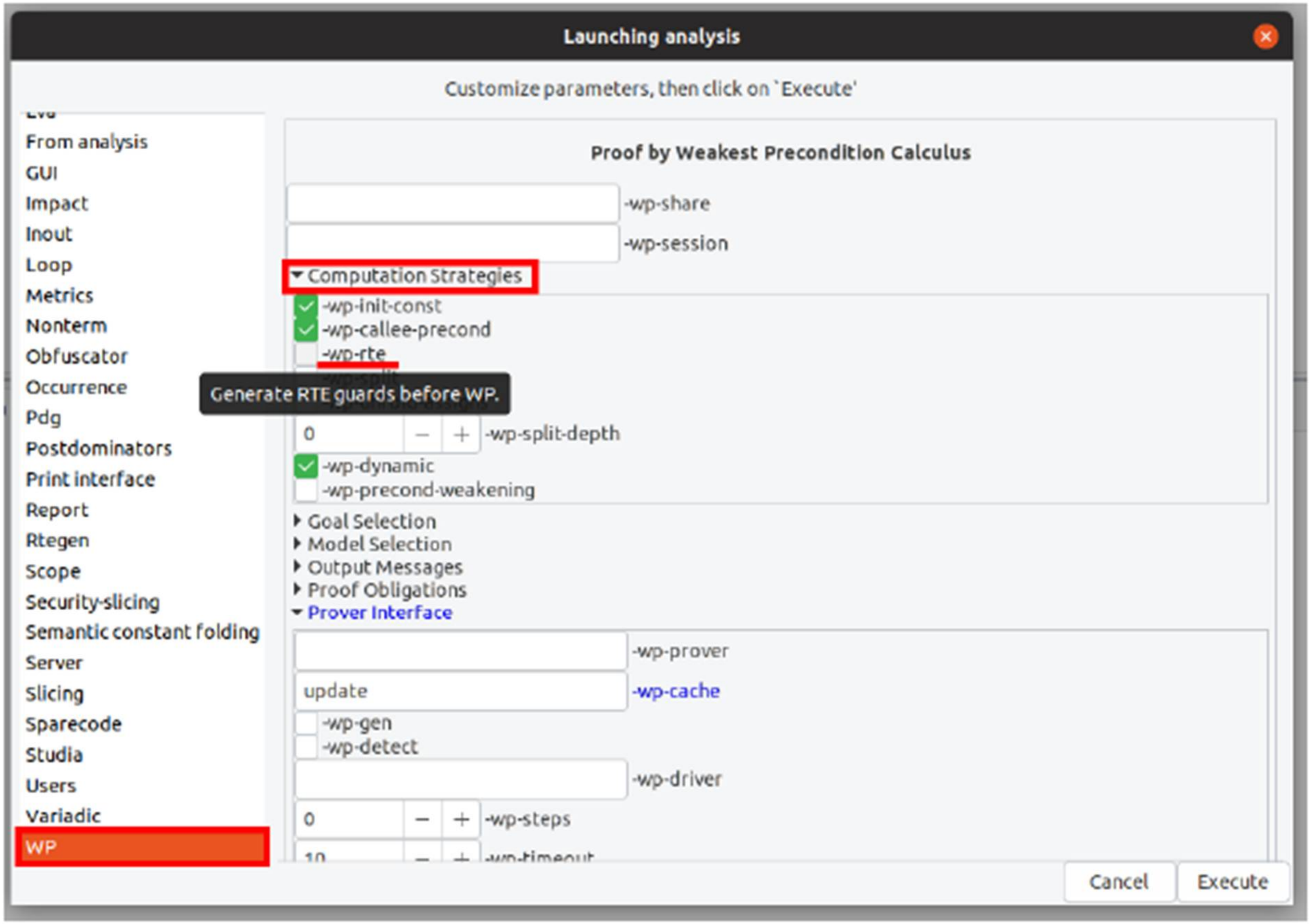


```
/*@ ensures \result ≥ 0;
    ensures
      (\old(val) ≥ 0 ⇒ \result ≡ \old(val)) ∧
      (\old(val) < 0 ⇒ \result ≡ -\old(val));
 */
int abs(int val)
{
  int __retres;
  if (val < 0) {
    {
      __retres =
      goto retur
    }
  }
  __retres = val;
  return_label: return __retres;
}
```

$ frama-c-gui
$ frama-c-gui abs.c

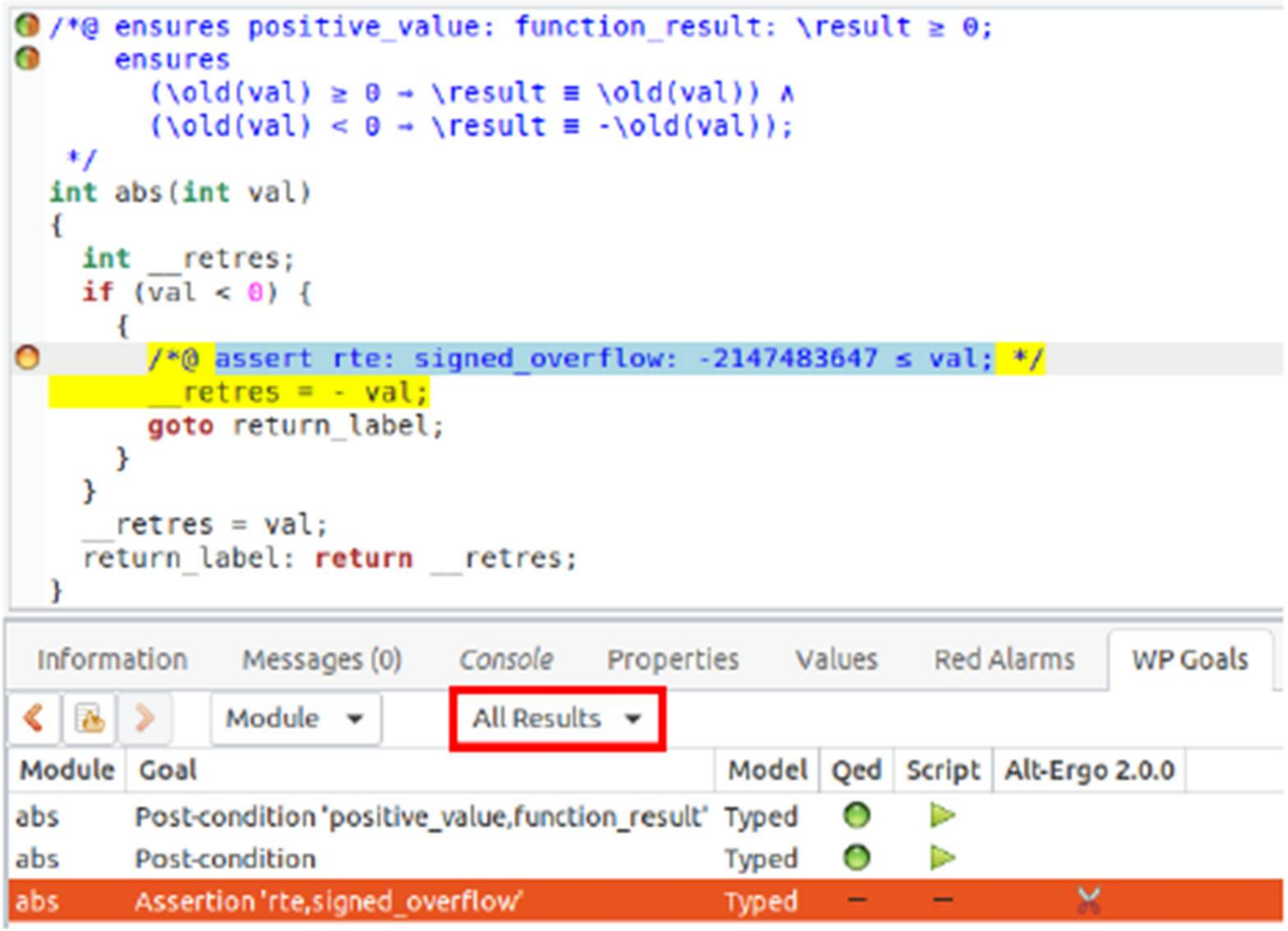


```
/*@ ensures positive_value: function_result: \result ≥ 0;
    ensures
      (\old(val) ≥ 0 → \result ≡ \old(val)) ∧
      (\old(val) < 0 → \result ≡ -\old(val));
 */
int abs(int val)
{
  int __retres;
  if (val < 0) {
    __retres = - val;
    goto return_label;
  }
  __retres = val;
  return_label: return __retres;
}
```



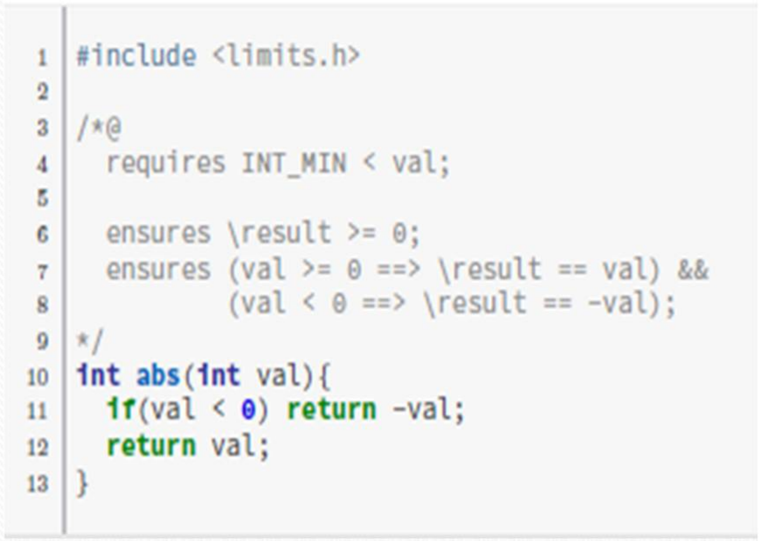
To activate -wp-rte always to verify examples.

click on the plugin configuration button: add the option -wp-rte in the options WP

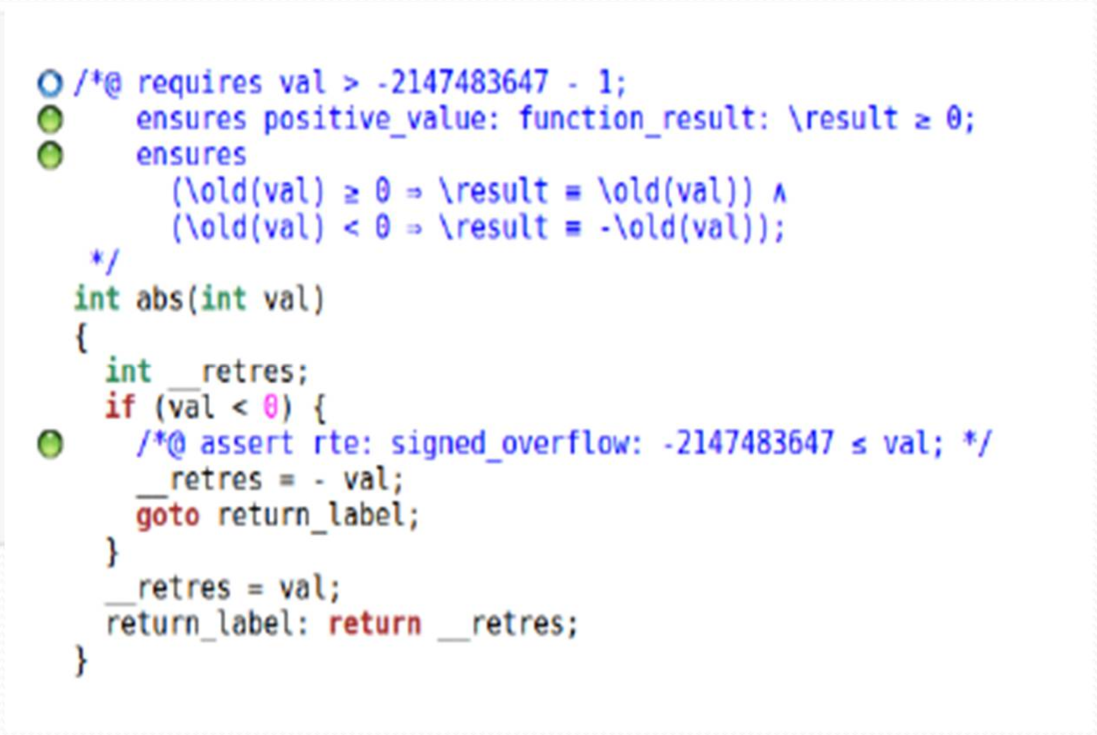


```
/*@ ensures positive_value: function_result: \result ≥ 0;
    ensures
      (\old(val) ≥ 0 → \result ≡ \old(val)) ∧
      (\old(val) < 0 → \result ≡ -\old(val));
 */
int abs(int val)
{
  int __retres;
  if (val < 0) {
    {
      /*@ assert rte: signed_overflow: -2147483647 ≤ val; */
      __retres = - val;
      goto return_label;
    }
  }
  __retres = val;
  return_label: return __retres;
}
```

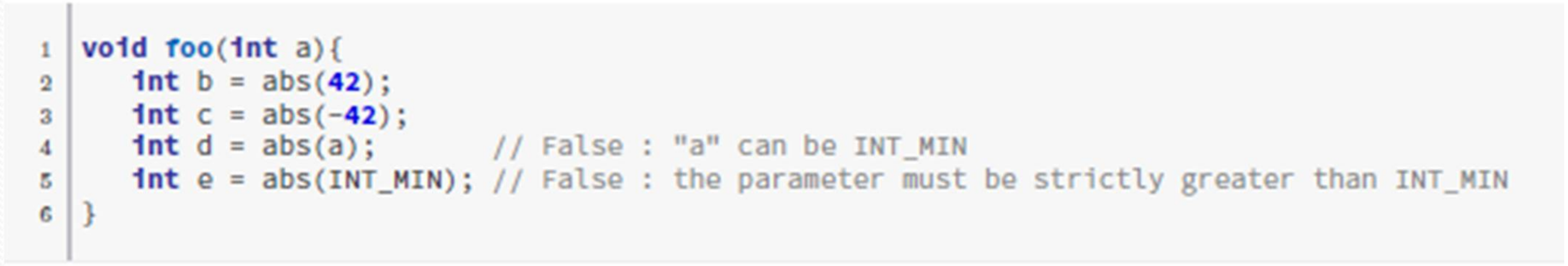
Module	Goal	Model	Qed	Script	Alt-Ergo 2.0.0
abs	Post-condition 'positive_value,function_result'	Typed	●	▷	
abs	Post-condition	Typed	●	▷	
abs	Assertion 'rte,signed_overflow'	Typed	–	–	✗



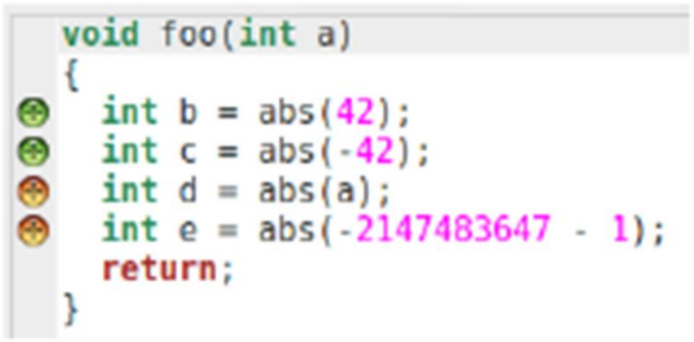
```
#include <limits.h>

/*@
  requires INT_MIN < val;

  ensures \result >= 0;
  ensures (val >= 0 ==> \result == val) &&
          (val < 0 ==> \result == -val);
*/
int abs(int val){
  if(val < 0) return -val;
  return val;
}
```



```
/*@ requires val > -2147483647 - 1;
    ensures positive_value: function_result: \result ≥ 0;
    ensures
      (\old(val) ≥ 0 ⇒ \result ≡ \old(val)) ∧
      (\old(val) < 0 ⇒ \result ≡ -\old(val));
 */
int abs(int val)
{
  int __retres;
  if (val < 0) {
    /*@ assert rte: signed_overflow: -2147483647 ≤ val; */
    __retres = - val;
    goto return_label;
  }
  __retres = val;
  return_label: return __retres;
}
```

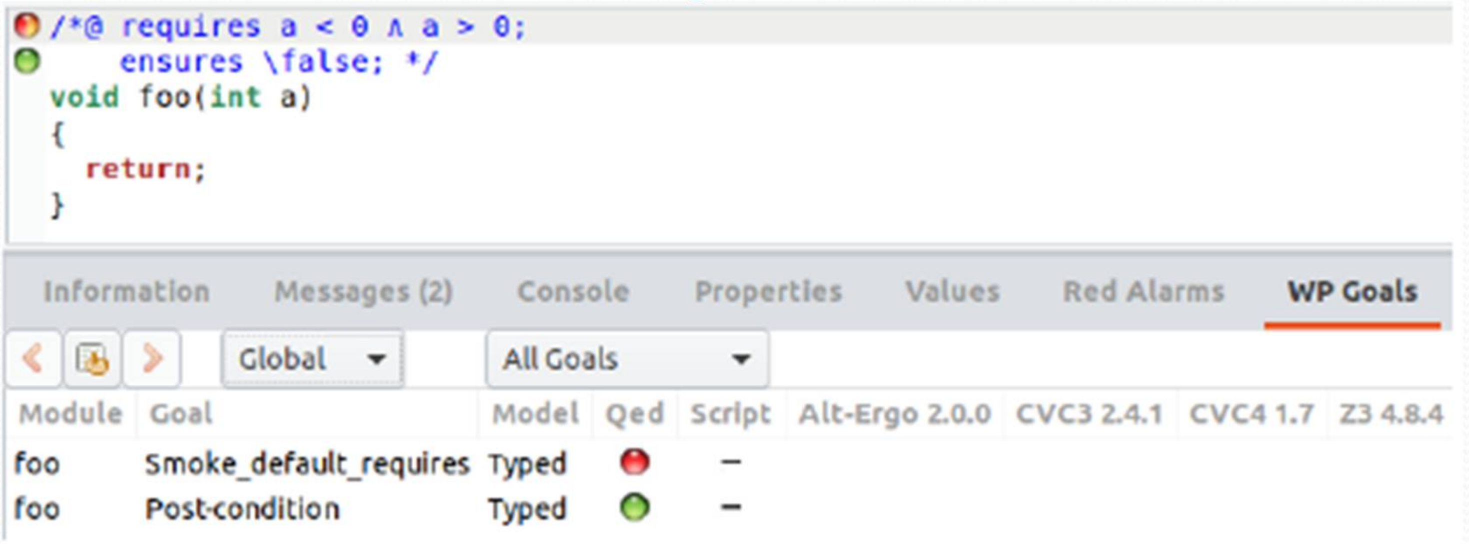


```
void foo(int a){
   int b = abs(42);
   int c = abs(-42);
   int d = abs(a);        // False : "a" can be INT_MIN
   int e = abs(INT_MIN); // False : the parameter must be strictly greater than INT_MIN
}
```



```
void foo(int a)
{
  int b = abs(42);
  int c = abs(-42);
  int d = abs(a);
  int e = abs(-2147483647 - 1);
  return;
}
```

Inconsistent preconditions

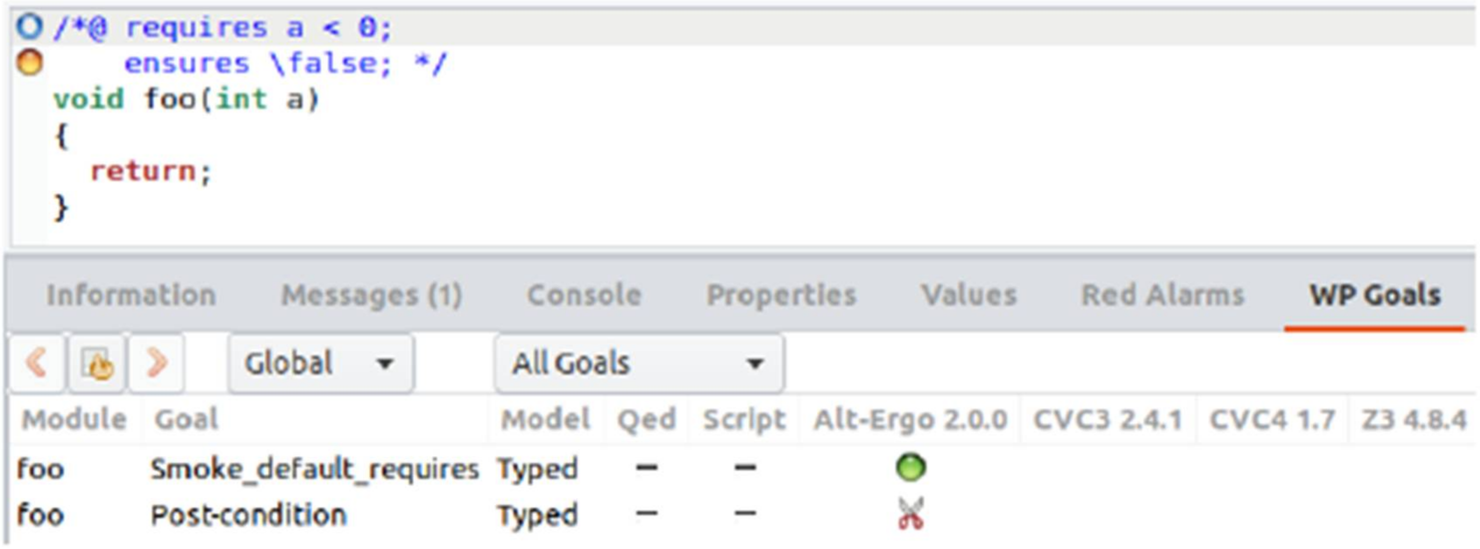


```
/*@ requires a < 0 ∧ a > 0;
    ensures \false; */
void foo(int a)
{
  return;
}
```

Information | Messages (2) | Console | Properties | Values | Red Alarms | WP Goals

Global | All Goals

Module	Goal	Model	Qed	Script	Alt-Ergo 2.0.0	CVC3 2.4.1	CVC4 1.7	Z3 4.8.4
foo	Smoke_default_requires	Typed	🔴	–				
foo	Post-condition	Typed	🟢	–				



```
/*@ requires a < 0;
    ensures \false; */
void foo(int a)
{
  return;
}
```

Information | Messages (1) | Console | Properties | Values | Red Alarms | WP Goals

Global | All Goals

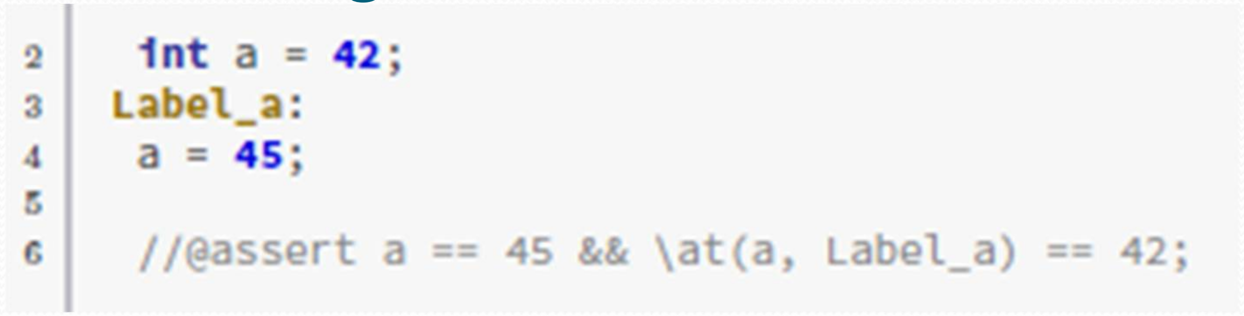
Module	Goal	Model	Qed	Script	Alt-Ergo 2.0.0	CVC3 2.4.1	CVC4 1.7	Z3 4.8.4
foo	Smoke_default_requires	Typed	–	–	🟢			
foo	Post-condition	Typed	–	–	✂			

Built-in labels

- 6 built-in labels defined by ACSL that can be used:
  - Pre / Old : value before function call,
  - Post : value after function call,
  - LoopEntry : value at loop entry
  - LoopCurrent : value at the beginning of the current step of the loop,
  - Here : value at the current program point.

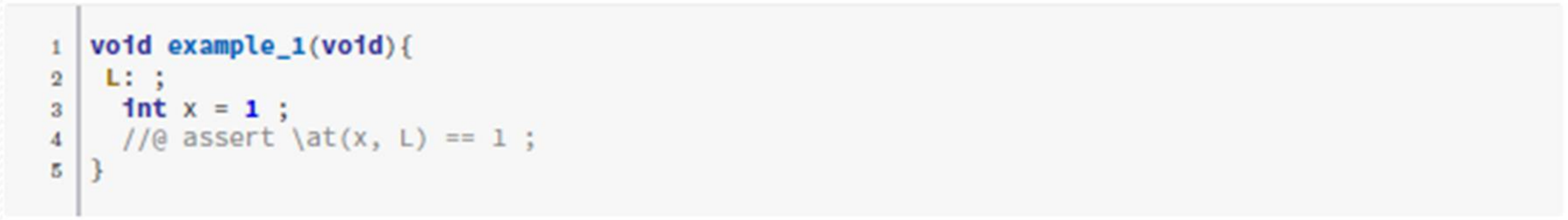
- \old can only be used in function postconditions, \at can be used anywhere.
- Old and Post are only available in function postconditions, Pre and Here are available everywhere.
- LoopEntry and LoopCurrent are only available in the context of loops

\at usage

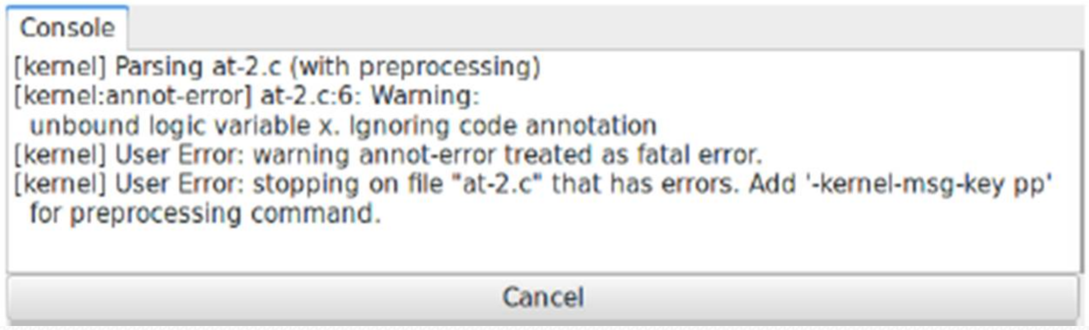


```
  int a = 42;
 Label_a:
  a = 45;

  //@assert a == 45 && \at(a, Label_a) == 42;
```

program, Frama-C detects that we ask the value of the variable `x` at a program point where it does not exist:

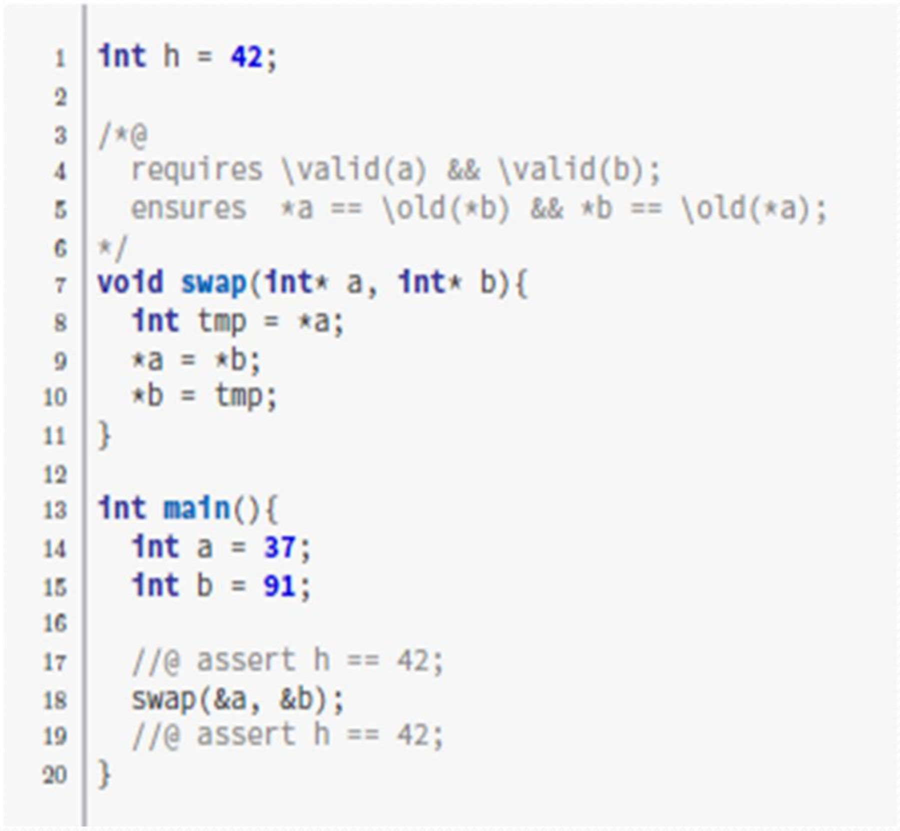


```
void example_1(void){
 L: ;
  int x = 1 ;
  //@ assert \at(x, L) == 1 ;
}
```

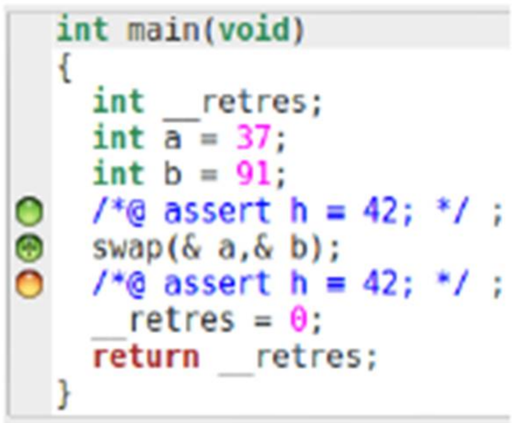


```
[kernel] Parsing at-2.c (with preprocessing)
[kernel:annot-error] at-2.c:6: Warning:
  unbound logic variable x. Ignoring code annotation
[kernel] User Error: warning annot-error treated as fatal error.
[kernel] User Error: stopping on file "at-2.c" that has errors. Add '-kernel-msg-key pp'
  for preprocessing command.
```

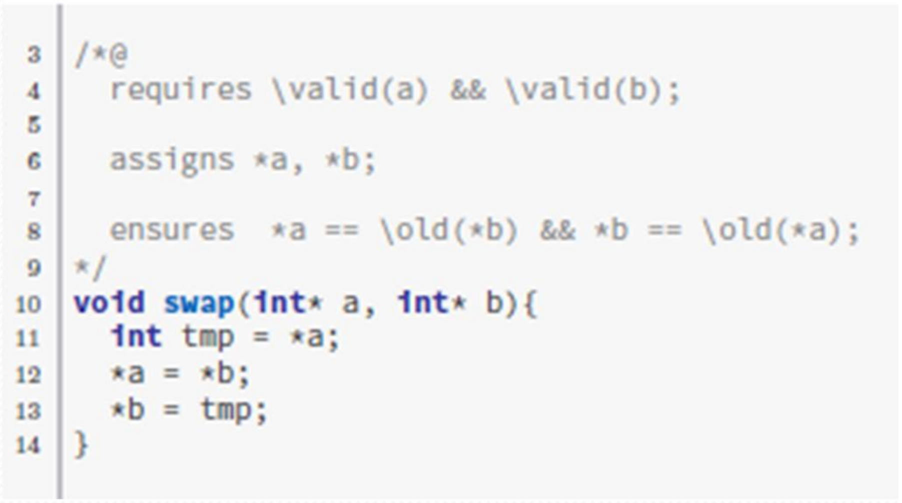
Need of assigns clause



```
int h = 42;

/*@
  requires \valid(a) && \valid(b);
  ensures  *a == \old(*b) && *b == \old(*a);
*/
void swap(int* a, int* b){
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main(){
  int a = 37;
  int b = 91;

  //@ assert h == 42;
  swap(&a, &b);
  //@ assert h == 42;
}
```

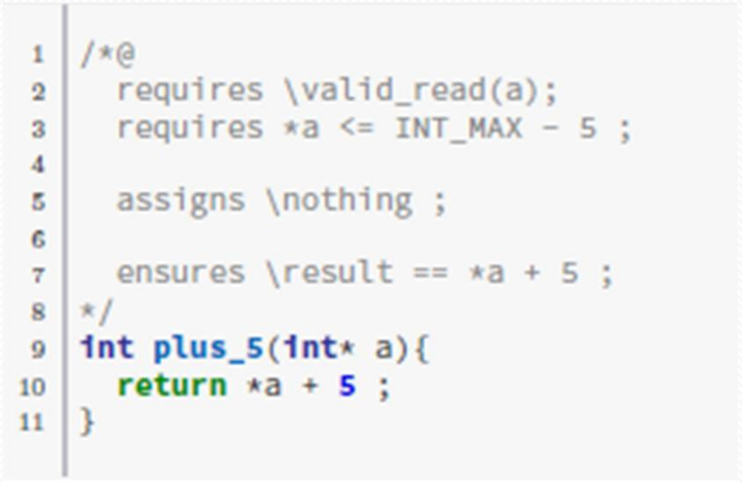


```
int main(void)
{
  int __retres;
  int a = 37;
  int b = 91;
  /*@ assert h = 42; */ ;
  swap(& a,& b);
  /*@ assert h = 42; */ ;
  __retres = 0;
  return __retres;
}
```



```
/*@
  requires \valid(a) && \valid(b);

  assigns *a, *b;

  ensures  *a == \old(*b) && *b == \old(*a);
*/
void swap(int* a, int* b){
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
```

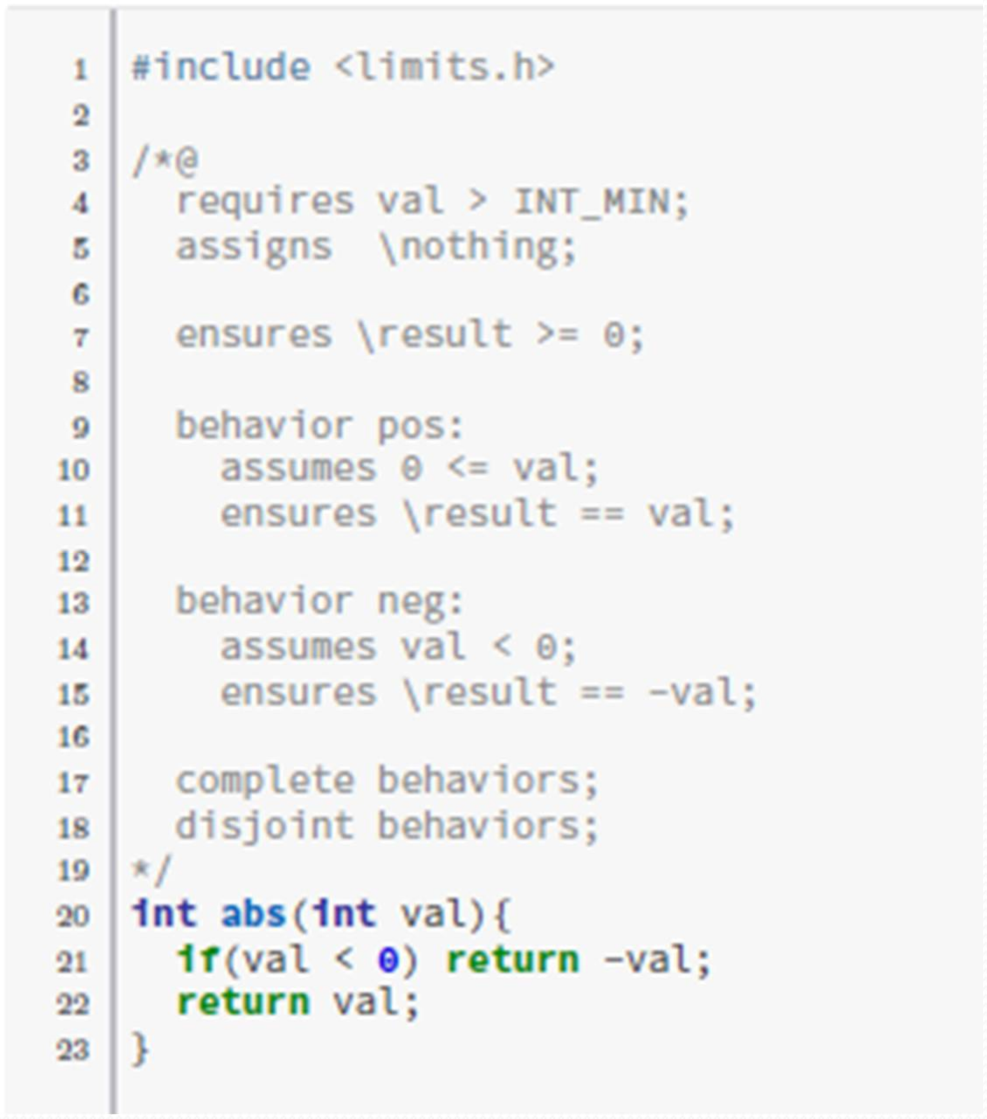
Pointer read only contract



```
/*@
  requires \valid_read(a);
  requires *a <= INT_MAX - 5 ;

  assigns \nothing ;

  ensures \result == *a + 5 ;
*/
int plus_5(int* a){
  return *a + 5 ;
}
```

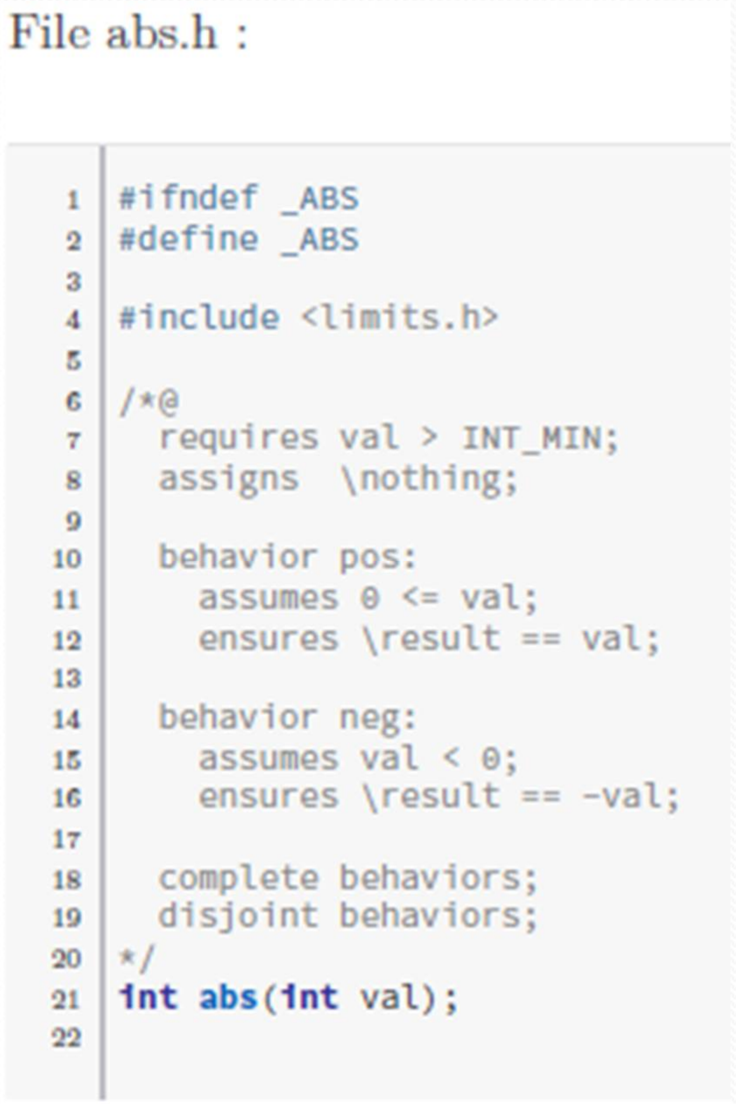
\valid_read indicates that a pointer can be dereferenced, but only to read the pointed memory

Behavior



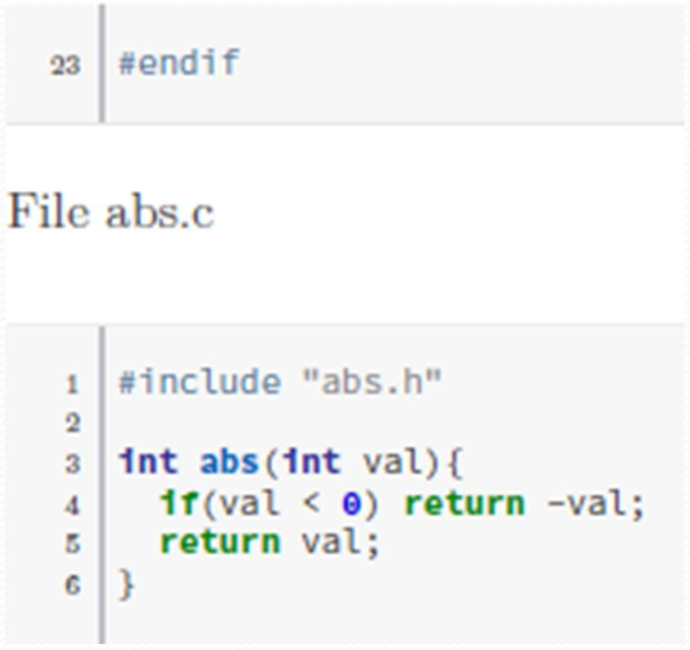
```
#include <limits.h>

/*@
  requires val > INT_MIN;
  assigns  \nothing;

  ensures \result >= 0;

  behavior pos:
    assumes 0 <= val;
    ensures \result == val;

  behavior neg:
    assumes val < 0;
    ensures \result == -val;

  complete behaviors;
  disjoint behaviors;
*/
int abs(int val){
  if(val < 0) return -val;
  return val;
}
```

Specification in Header file



File abs.h :

```
#ifndef _ABS
#define _ABS

#include <limits.h>

/*@
  requires val > INT_MIN;
  assigns  \nothing;

  behavior pos:
    assumes 0 <= val;
    ensures \result == val;

  behavior neg:
    assumes val < 0;
    ensures \result == -val;

  complete behaviors;
  disjoint behaviors;
*/
int abs(int val);

```

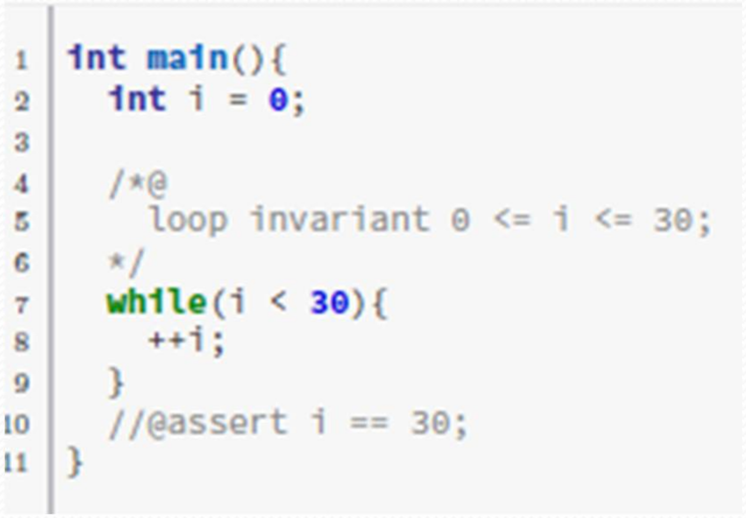


```
#endif
```

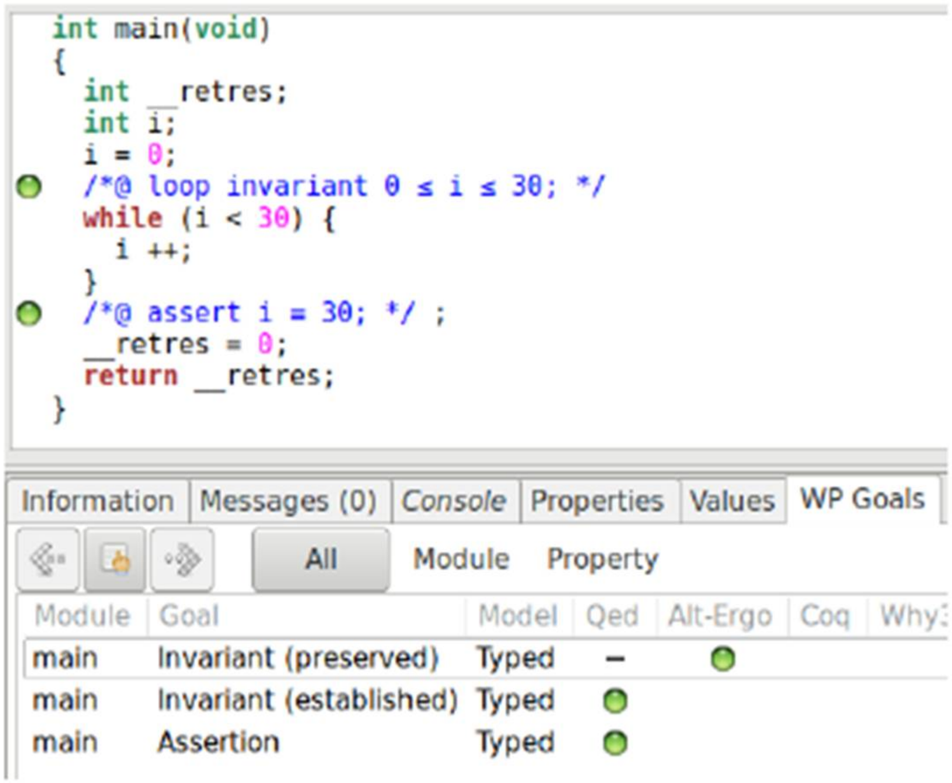
File abs.c

```
#include "abs.h"

int abs(int val){
  if(val < 0) return -val;
  return val;
}
```

Loop invariants



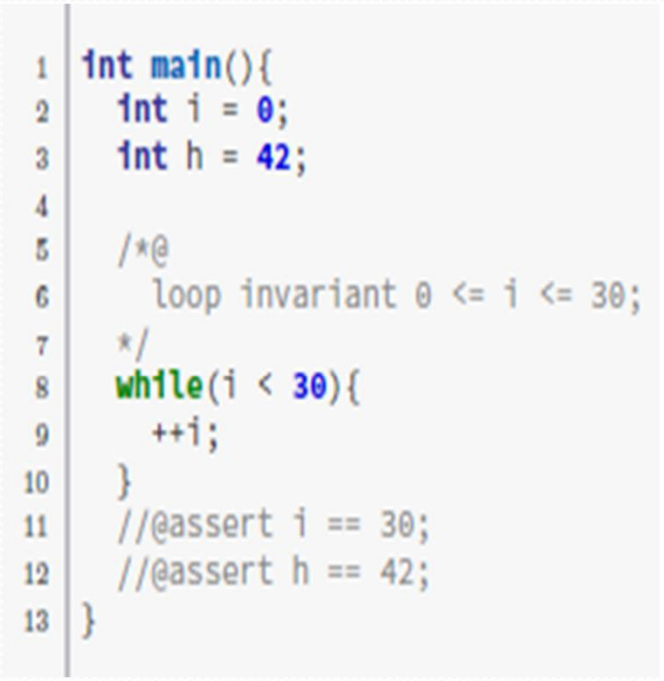
```
int main(){
  int i = 0;

  /*@
    loop invariant 0 <= i <= 30;
  */
  while(i < 30){
    ++i;
  }
  //@assert i == 30;
}
```



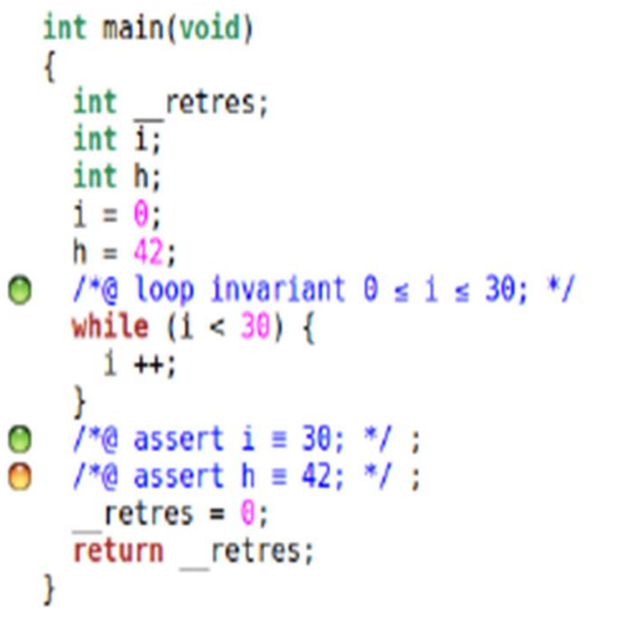
```
int main(void)
{
  int __retres;
  int i;
  i = 0;
  /*@ loop invariant 0 ≤ i ≤ 30; */
  while (i < 30) {
    i ++;
  }
  /*@ assert i ≡ 30; */ ;
  __retres = 0;
  return __retres;
}
```

Module	Goal	Model	Qed	Alt-Ergo	Coq	Why3
main	Invariant (preserved)	Typed	–	●		
main	Invariant (established)	Typed	●			
main	Assertion	Typed	●			

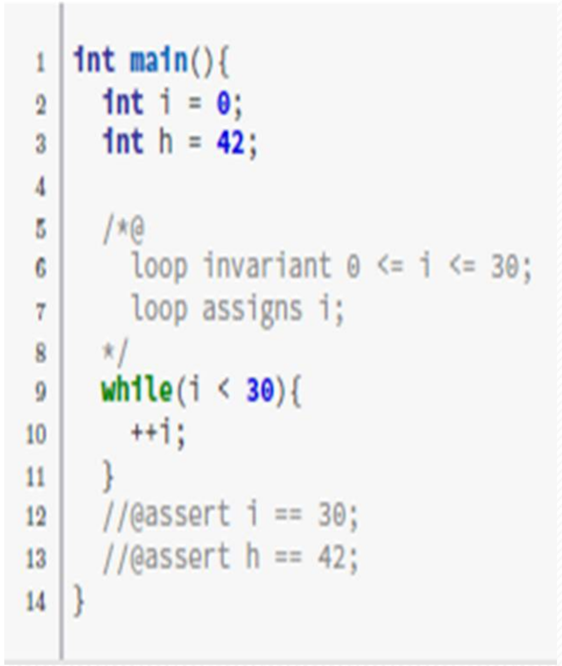
Loop assigns



```c
int main(){
  int i = 0;
  int h = 42;

  /*@
    loop invariant 0 <= i <= 30;
  */
  while(i < 30){
    ++i;
  }
  //@assert i == 30;
  //@assert h == 42;
}
```

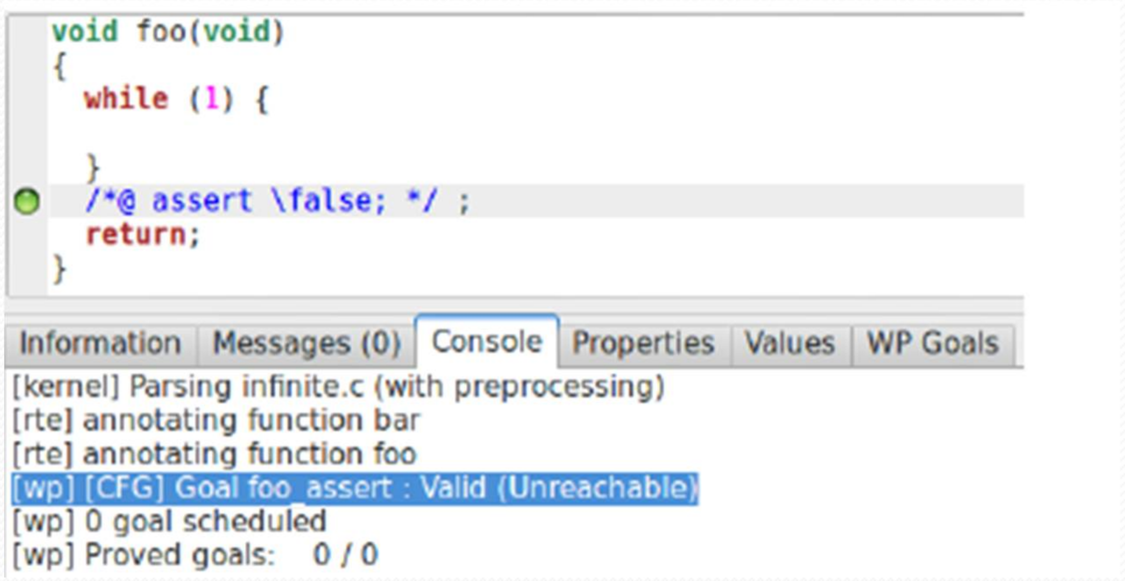


```c
int main(void)
{
  int __retres;
  int i;
  int h;
  i = 0;
  h = 42;
● /*@ loop invariant 0 ≤ i ≤ 30; */
  while (i < 30) {
    i ++;
  }
● /*@ assert i ≡ 30; */ ;
● /*@ assert h ≡ 42; */ ;
  __retres = 0;
  return __retres;
}
```



```c
int main(){
  int i = 0;
  int h = 42;

  /*@
    loop invariant 0 <= i <= 30;
    loop assigns i;
  */
  while(i < 30){
    ++i;
  }
  //@assert i == 30;
  //@assert h == 42;
}
```

Total correctness

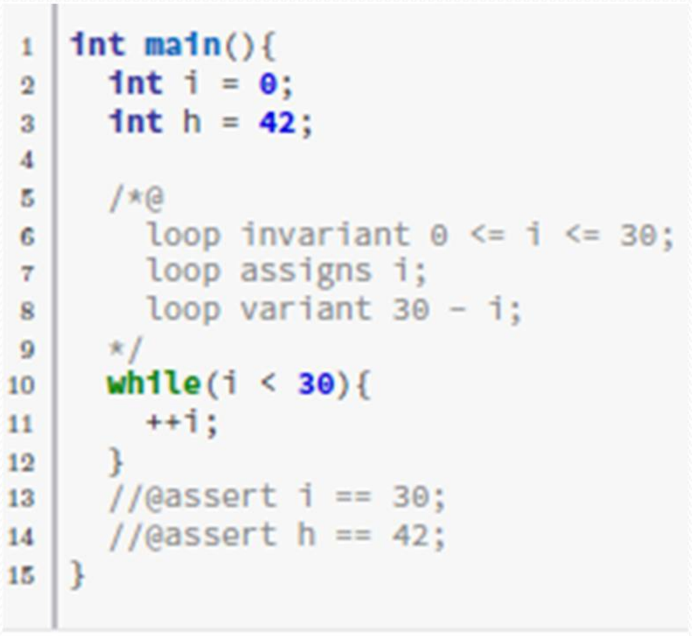


```
void foo(void)
{
  while (1) {

  }
  /*@ assert \false; */ ;
  return;
}
```

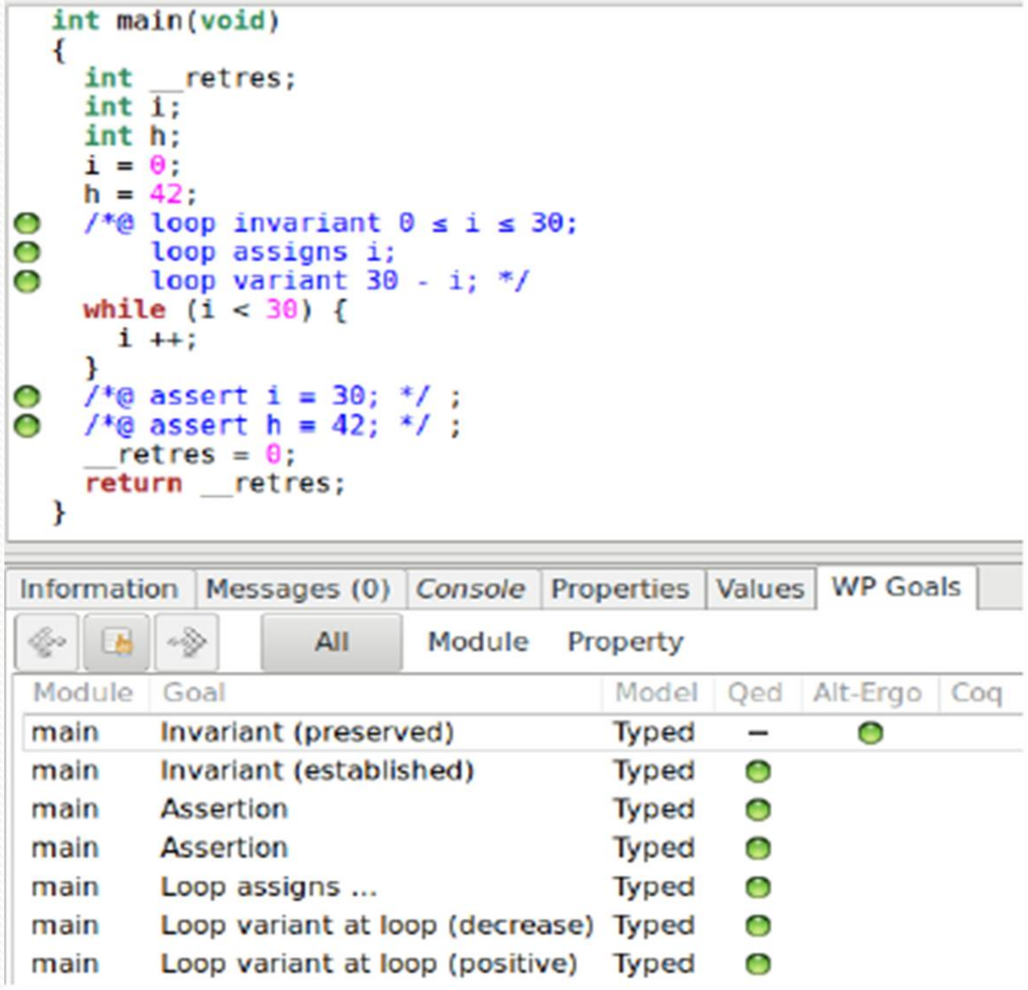
Information | Messages (0) | Console | Properties | Values | WP Goals

```
[kernel] Parsing infinite.c (with preprocessing)
[rte] annotating function bar
[rte] annotating function foo
[wp] [CFG] Goal foo_assert : Valid (Unreachable)
[wp] 0 goal scheduled
[wp] Proved goals:    0 / 0
```

Total correctness



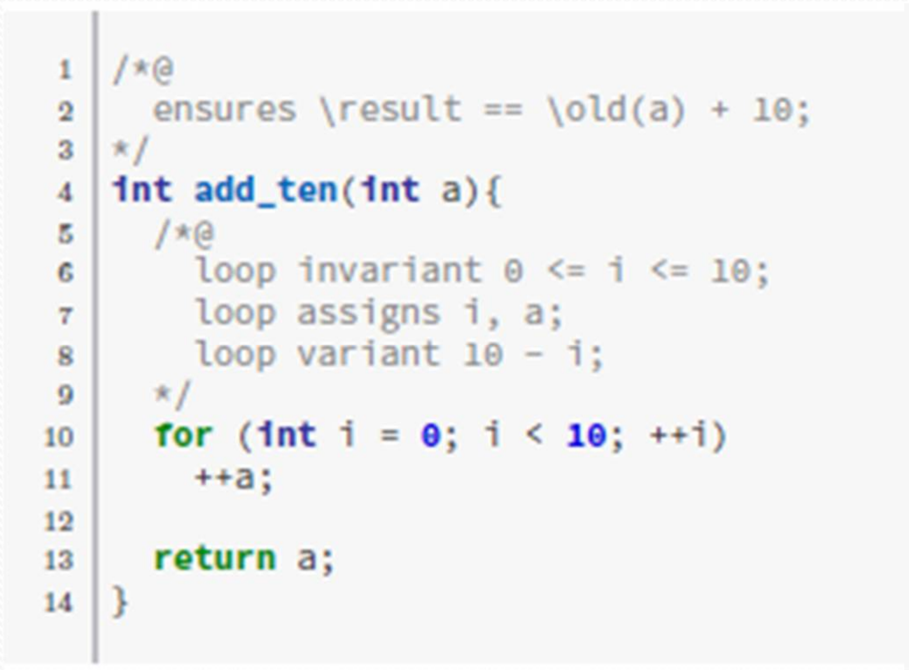
```
int main(){
  int i = 0;
  int h = 42;

  /*@
    loop invariant 0 <= i <= 30;
    loop assigns i;
    loop variant 30 - i;
  */
  while(i < 30){
    ++i;
  }
  //@assert i == 30;
  //@assert h == 42;
}
```



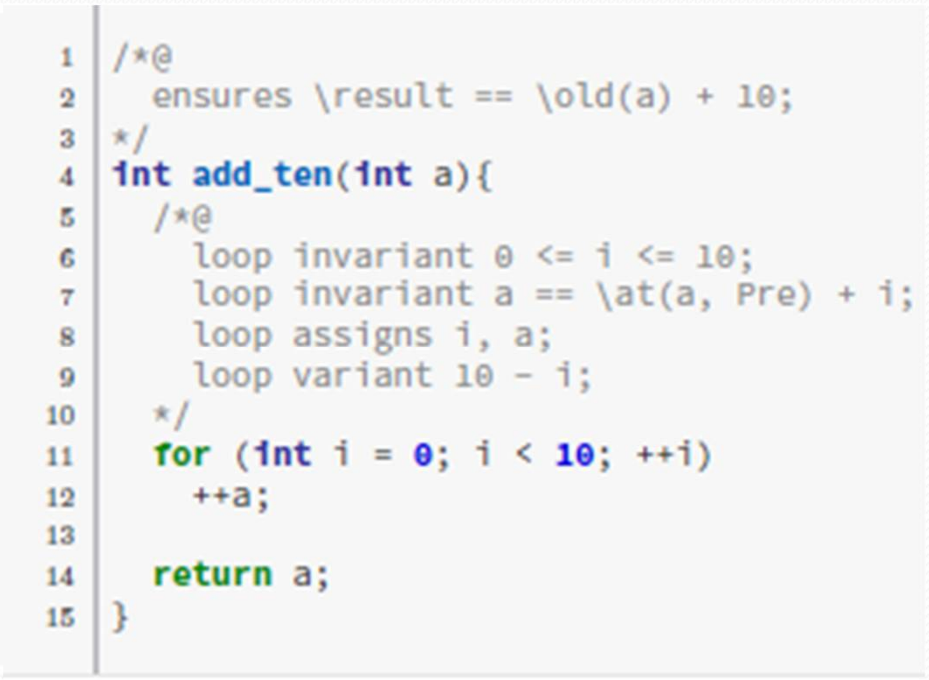
```
int main(void)
{
  int __retres;
  int i;
  int h;
  i = 0;
  h = 42;
● /*@ loop invariant 0 ≤ i ≤ 30;
●       loop assigns i;
●       loop variant 30 - i; */
  while (i < 30) {
    i ++;
  }
● /*@ assert i ≡ 30; */ ;
● /*@ assert h ≡ 42; */ ;
  __retres = 0;
  return __retres;
}
```

Module	Goal	Model	Qed	Alt-Ergo	Coq
main	Invariant (preserved)	Typed	–	●	
main	Invariant (established)	Typed	●		
main	Assertion	Typed	●		
main	Assertion	Typed	●		
main	Loop assigns ...	Typed	●		
main	Loop variant at loop (decrease)	Typed	●		
main	Loop variant at loop (positive)	Typed	●		

Post condition with invariant

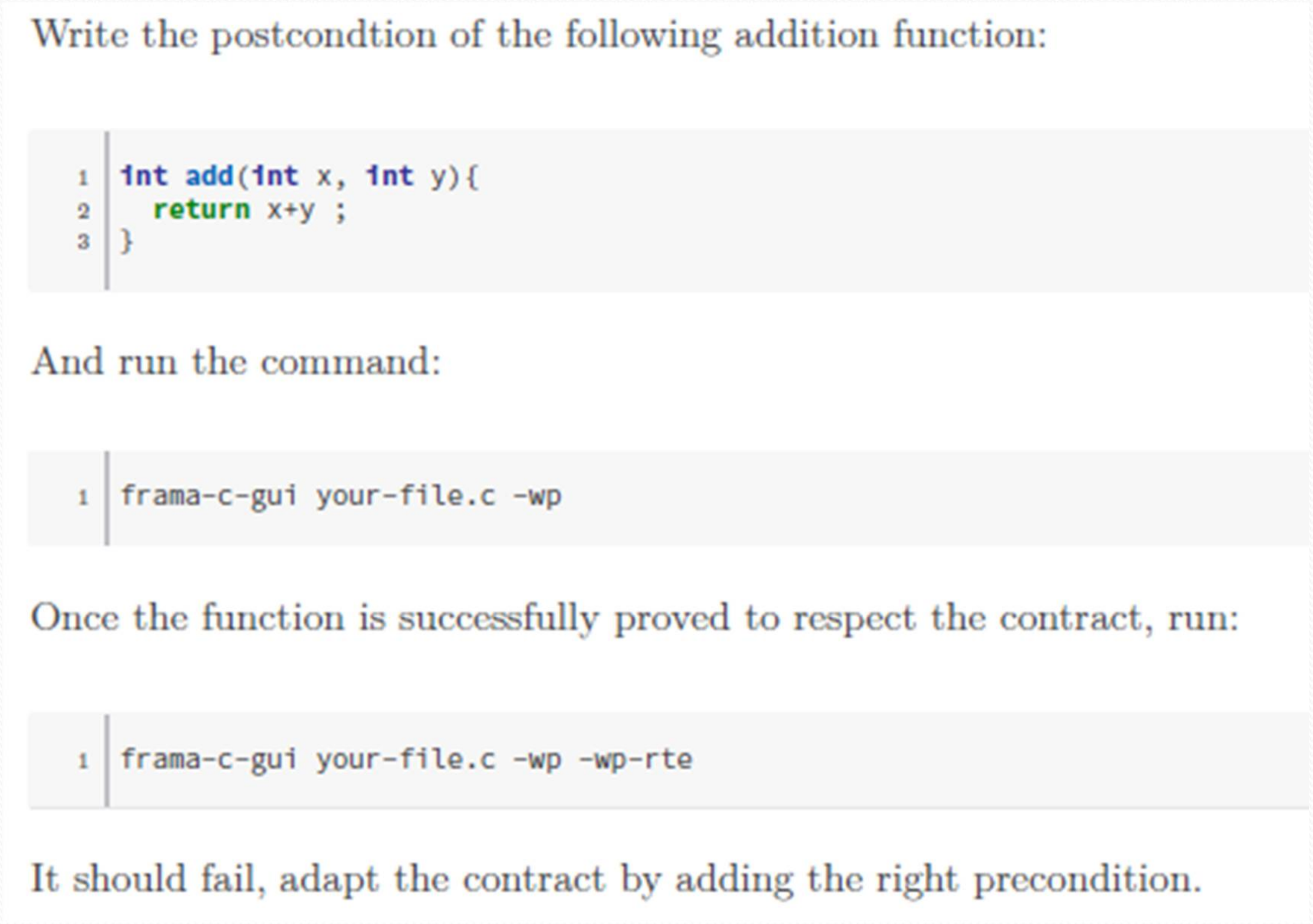


```c
/*@
  ensures \result == \old(a) + 10;
*/
int add_ten(int a){
  /*@
    loop invariant 0 <= i <= 10;
    loop assigns i, a;
    loop variant 10 - i;
  */
  for (int i = 0; i < 10; ++i)
    ++a;

  return a;
}
```



```c
/*@
  ensures \result == \old(a) + 10;
*/
int add_ten(int a){
  /*@
    loop invariant 0 <= i <= 10;
    loop invariant a == \at(a, Pre) + i;
    loop assigns i, a;
    loop variant 10 - i;
  */
  for (int i = 0; i < 10; ++i)
    ++a;

  return a;
}
```

Exercise :add



Write the postcondtion of the following addition function:

```c
int add(int x, int y){
  return x+y ;
}
```

And run the command:

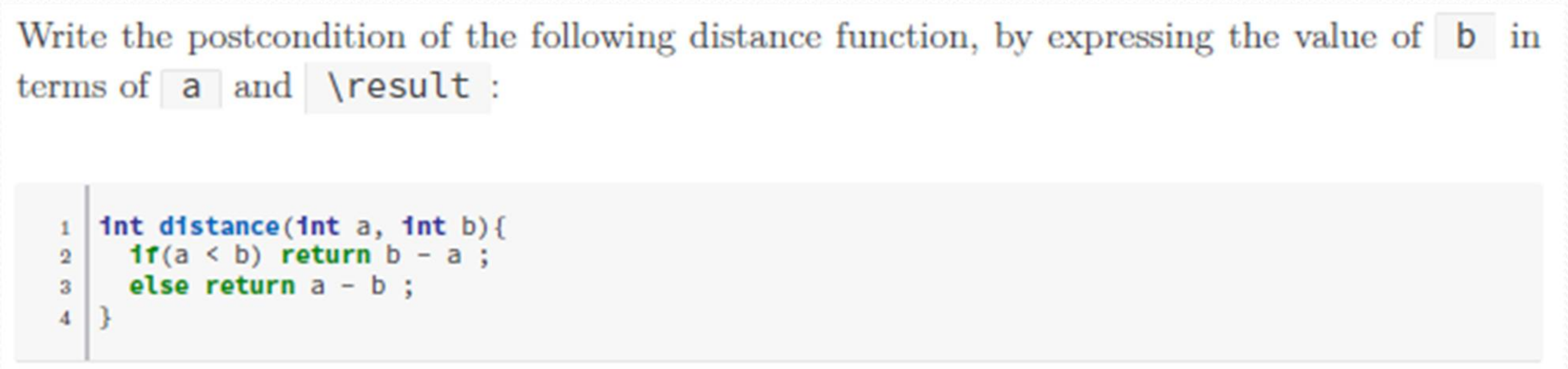
```
frama-c-gui your-file.c -wp
```

Once the function is successfully proved to respect the contract, run:

```
frama-c-gui your-file.c -wp -wp-rte
```

It should fail, adapt the contract by adding the right precondition.

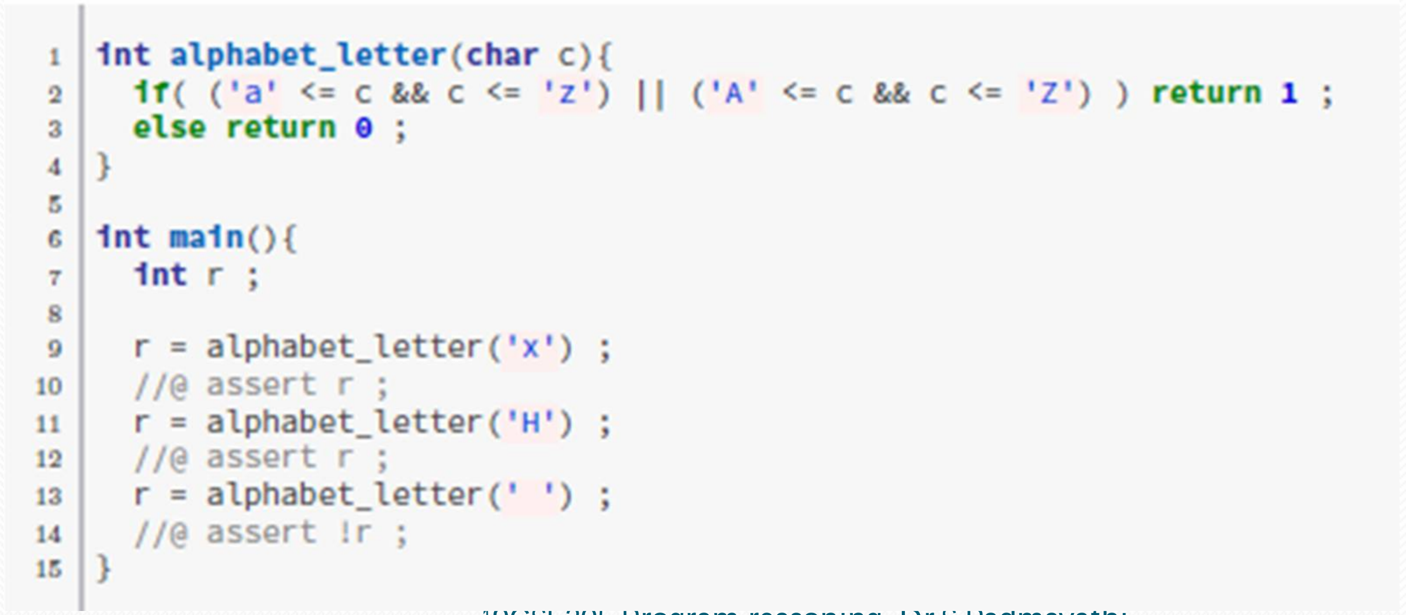
Exercise: distance, character



Write the postcondition of the following distance function, by expressing the value of `b` in terms of `a` and `\result` :

```
int distance(int a, int b){
  if(a < b) return b - a ;
  else return a - b ;
}
```

Write the postcondition of the following function that return true if the character received in input is an alphabet letter. Use the equivalence operator <==> .



```
int alphabet_letter(char c){
  if( ('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z') ) return 1 ;
  else return 0 ;
}

int main(){
  int r ;

  r = alphabet_letter('x') ;
  //@ assert r ;
  r = alphabet_letter('H') ;
  //@ assert r ;
  r = alphabet_letter(' ') ;
  //@ assert !r ;
}
```

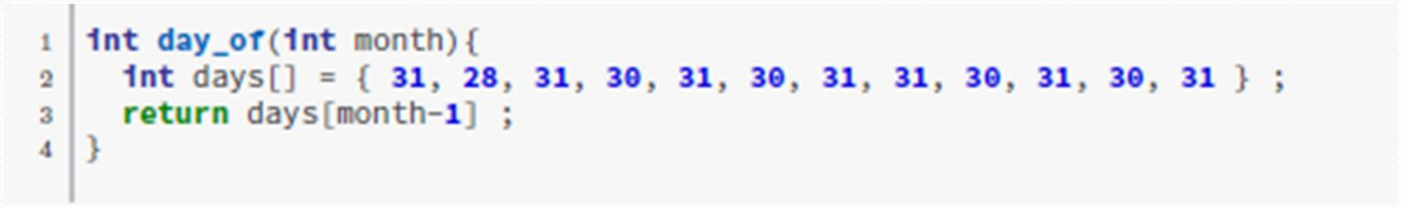
Exercise: month days

Write the postcondition of the following function that returns the number of days in function of the received month

ACSL provide the notion of set, and the operator \in that can be used to check whether a value is in a set or not.

For example:
1 //@ assert 13 \in { 1, 2, 3, 4, 5 } ; // FALSE
2 //@ assert 3 \in { 1, 2, 3, 4, 5 } ; // TRUE

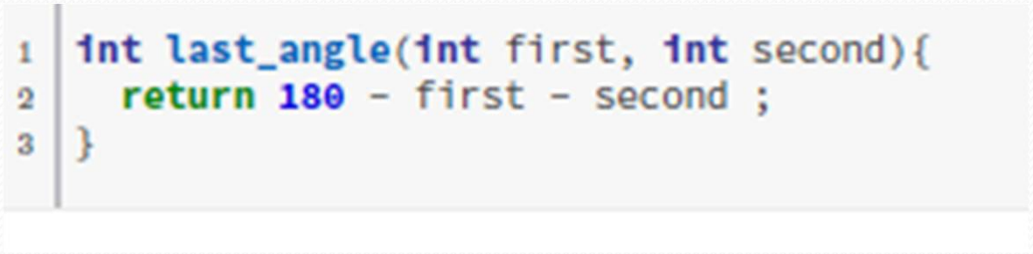
Modify the postcondition by using this notation.



```
int day_of(int month){
  int days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 } ;
  return days[month-1] ;
}
```

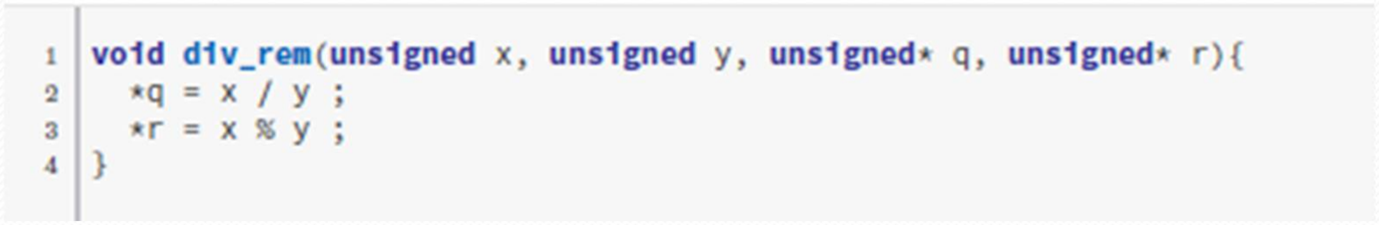
Exercise: triangle, digit extraction

This function receives two values of angle in input and returns the value of the last angle considering that the sum of the three angles must be 180. Write the postcondition that expresses that the sum of the three angles is 180.



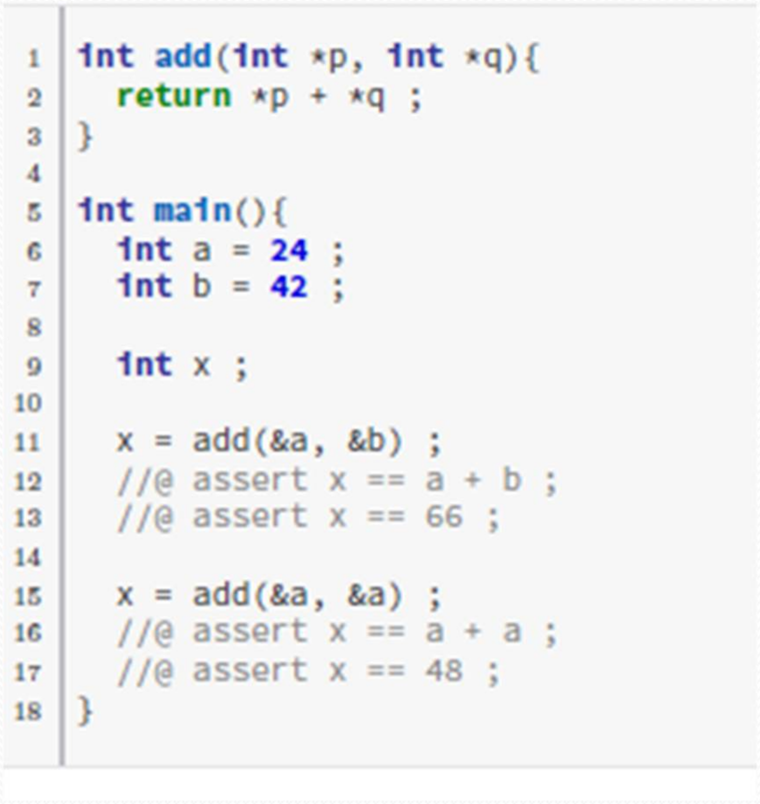
```
int last_angle(int first, int second){
  return 180 - first - second ;
}
```

Specify the postcondition of the following function, that computes the results of the division of a by b and its remaining and stores it into two memory locations p and q

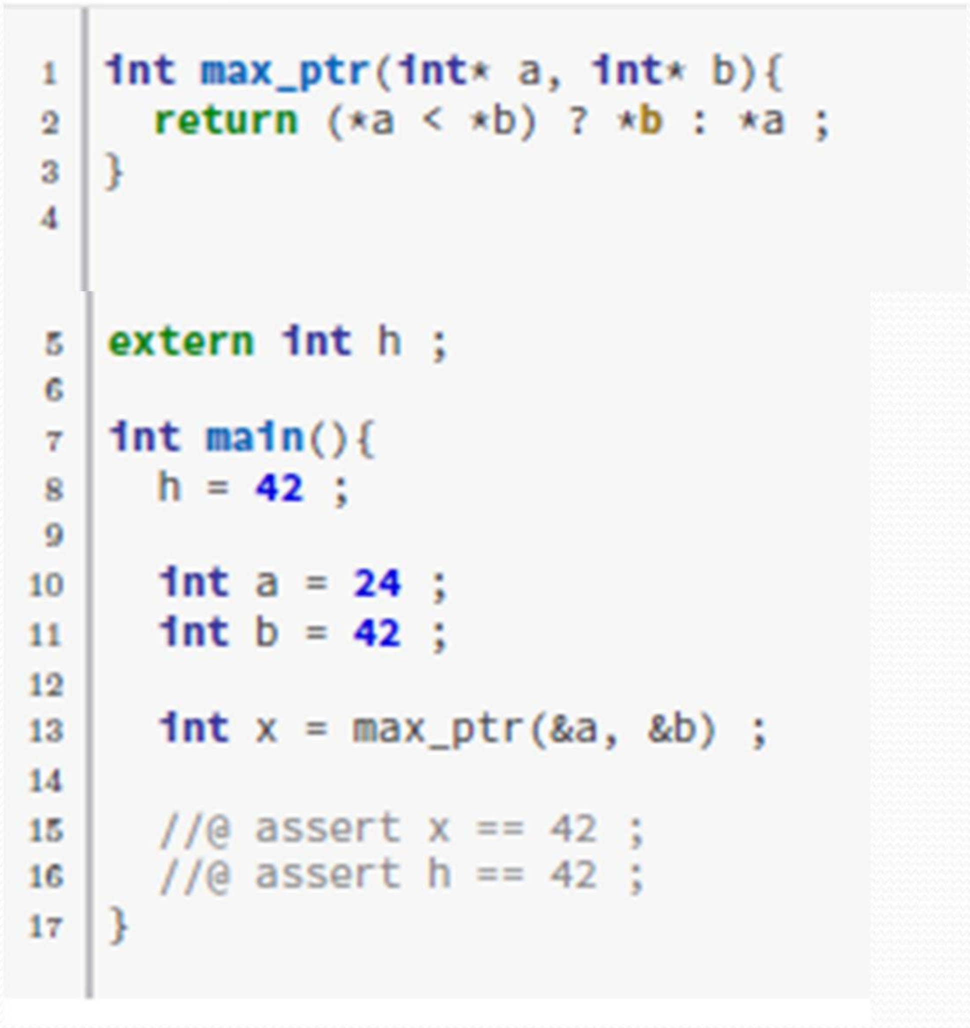


```
void div_rem(unsigned x, unsigned y, unsigned* q, unsigned* r){
  *q = x / y ;
  *r = x % y ;
}
```

Exercise: Pointer add, max



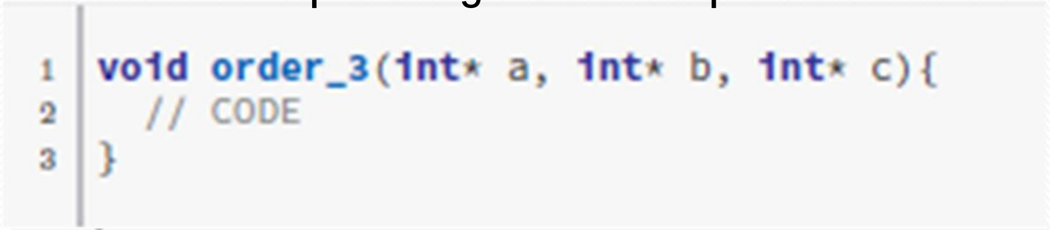
```c
int add(int *p, int *q){
  return *p + *q ;
}

int main(){
  int a = 24 ;
  int b = 42 ;

  int x ;

  x = add(&a, &b) ;
  //@ assert x == a + b ;
  //@ assert x == 66 ;

  x = add(&a, &a) ;
  //@ assert x == a + a ;
  //@ assert x == 48 ;
}
```



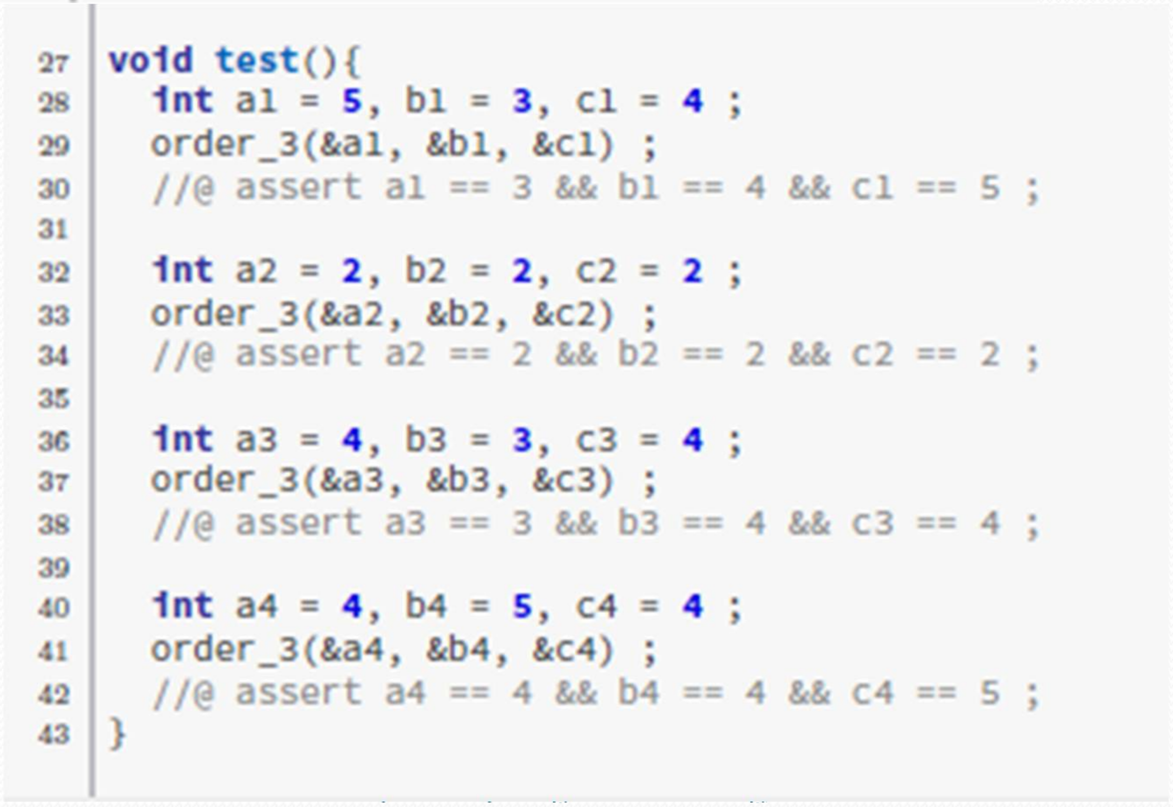
```c
int max_ptr(int* a, int* b){
  return (*a < *b) ? *b : *a ;
}

extern int h ;

int main(){
  h = 42 ;

  int a = 24 ;
  int b = 42 ;

  int x = max_ptr(&a, &b) ;

  //@ assert x == 42 ;
  //@ assert h == 42 ;
}
```

Exercise: order 3 numbers

function should order the 3 input values in increasing order.
Write the corresponding code and specification of the function



```c
void order_3(int* a, int* b, int* c){
  // CODE
}
```



```c
void test(){
  int a1 = 5, b1 = 3, c1 = 4 ;
  order_3(&a1, &b1, &c1) ;
  //@ assert a1 == 3 && b1 == 4 && c1 == 5 ;

  int a2 = 2, b2 = 2, c2 = 2 ;
  order_3(&a2, &b2, &c2) ;
  //@ assert a2 == 2 && b2 == 2 && c2 == 2 ;

  int a3 = 4, b3 = 3, c3 = 4 ;
  order_3(&a3, &b3, &c3) ;
  //@ assert a3 == 3 && b3 == 4 && c3 == 4 ;

  int a4 = 4, b4 = 5, c4 = 4 ;
  order_3(&a4, &b4, &c4) ;
  //@ assert a4 == 4 && b4 == 4 && c4 == 5 ;
}
```