14.

#include<limits.h>

/\*@

requires val>INT\_MIN;

assigns \nothing;

behavior pos:

assumes val>0;

ensures \result==val;

behavior neg:

assumes val<0;

ensures \result==-val;

behavior eq:

assumes val==0;

ensures \result==0;

complete behaviors;

disjoint behaviors;

\*/

int check(int val){

if(val>0)

return val;

else if(val<0)

return -val;

else

return 0;

}

int main(){

int x=4;

int r=check(x);

//@assert x==4;

}

15.

#include<limits.h>

/\*@

requires \valid(array+(0..length-1));

assigns array[0..length-1];

ensures \forall size\_t i; 0<=i<length && \old(array[i] == old ==> array[i] == new;

ensures \forall size\_t i; 0<=i<length && \old(array[i]) != old ==> array[i] == \old(array[i]);

\*/

void search\_and\_replace(int \*array,size\_t length,int old,int new){

/\*@

loop invariant 0<=i<=length;

loop invariant \forall size\_t j; 0<=j<i && \at(array[j],Pre) == old ==> array[i] == new;

loop invariant \forall size\_t j; 0<=j<i && \at(array[j],Pre) != old ==> array[j] == \at(array[i],Pre);

loop assigns i,array[0..length-1];

loop variant length-i;

\*/

for(size\_t i=0;i<length;++i){

if(array[i] == old)

array[i] = new;

}

}

16.

#include <stddef.h>

/\*@requires Sorted:

\forall integer i, j;0<=i<=j<len==>arr[i]<=arr[j];

\*/

int bsearch(int\* arr, int len, int value)

{

if(len==0)return -1;

int low=0;

int up=len-1;

while(low<=up){

int mid=low+(up-low)/2;

if(arr[mid]>value)up=mid-1;

else if(arr[mid]<value)low=mid+1;

else return mid;

}

return -1;

}

18.

#include <stddef.h>

/\*@

requires \valid\_read(a + (0..n-1));

assigns \nothing;

behavior empty:

assumes n == 0;

ensures \result == 0;

behavior not\_empty:

assumes 0 < n;

ensures 0 <= \result < n;

ensures \forall integer i; 0 <= i < n ==> a[i] <= a[\result];

ensures \forall integer i; 0 <= i < \result ==> a[i] < a[\result];

complete behaviors;

disjoint behaviors;

\*/

size\_t max\_element(const \* a, size\_t n)

{

if (n == 0) {

return 0;

}

size\_t max = 0;

/\*@

loop invariant 0 <= i <= n;

loop invariant 0 <= max < n;

loop invariant \forall integer k; 0 <= k < i ==> a[k] <= a[max];

loop invariant \forall integer k; 0 <= k < max ==> a[k] < a[max];

loop assigns max, i;

loop variant n-i;

\*/

for (size\_t i = 1; i < n; i++)

{

if (a[max] < a[i]) {

max = i;

}

}

return max;

}

int main()

{

int Z[5]={1,2,3,4,5};

max\_element(Z,3);

}

19.

#include <stddef.h>

#include<stdio.h>

/\*@

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;

}

/\*@

requires \valid(array + (0 .. len-1)) ;

assigns array[0 .. len-1];

ensures \forall integer j ; 0 <= j < len ==> array[j] == \old(array[len-j-1]);

\*/

void reverse(int\* array, size\_t len){

/\*@

loop invariant 0 <= i <= len/2 ;

loop invariant

\forall integer j ; (0 <= j < i || len-i <= j < len) ==>

array[j] == \at(array[len-j-1], Pre);

loop invariant

\forall integer j ; i <= j < len-i ==> array[j] == \at(array[j], Pre);

loop assigns i, array[0 .. len-1] ;

loop variant len-i ;

\*/

for(size\_t i = 0 ; i < len/2 ; ++i){

swap(array+i, array+len-i-1) ;

  }

}

20.

#include<stddef.h>

/\*@

requires \valid(array + (0..length-1));

assigns array[0..length-1];

ensures \forall size\_t i;0<=i<length && array[i]==\old(array[length-i]);

\*/

int reverse\_array(int\* array,size\_t length){

/\*@

loop invariant length-1<=i<=0;

loop invariant \forall size\_t i;0<=i<length && array[i]==\old(array[length-i]);

loop assigns i,array[0..length-1];

loop variant length-i;

\*/

for(size\_t i=length;i>0;i--){

return array[length-i];

}

}

int main(){

int a[]={1,2,3,4,5};

int r=reverse\_array(&a,5);

return 0;

}

21.

#include<stddef.h>

/\*@

requires \valid(array + (0..length-1));

assigns array[0..length-1];

ensures \forall size\_t i;0<=i<length && \old(array[i])==old ==> array[i] == new;

ensures \forall size\_t i;0<=i<length && \old(array[i])!=old ==> array[i] == \old(array[i]);

\*/

void search\_and\_replace(int\* array,size\_t length,int old,int new){

/\*@

loop invariant 0<=i<=length;

loop invariant \forall size\_t j;0<=j<i && \at(array[j],Pre)==old ==> array[j] == new;

loop invariant \forall size\_t j;0<=j<i && \at(array[j],Pre)!=old ==> array[j] == \at(array[j],Pre);

loop assigns i,array[0..length-1];

loop variant length-i;

\*/

for(size\_t i=0;i<length;i++){

if(array[i]==old) array[i]=new;

}

}

int main(){

int a[]={1,2,3,4,5};

search\_and\_replace(&a,5,4,6);

return 0;

}

22.

#include<stddef.h>

/\*@

requires \valid(array + (0..length-1));

assigns array[0..length-1];

ensures \forall size\_t i; 0 <= i < length ==> array[i] == 0;

\*/

void reset(int\* array,size\_t length){

/\*@

loop invariant 0 <= i <=length;

loop invariant \forall size\_t j; 0 <= j < i ==> array[j] == 0;

loop assigns i,array[0..length-1];

loop variant length-i;

\*/

for(size\_t i=0;i<length;i++){

array[i]=0;}

}

int main(){

int a[]={1,2,3,4,5};

reset(a,5);

}

ARRAY Q)

#include<stddef.h>

/\*@

requires \valid\_read(array + (0..length-1));

assigns \nothing;

behavior in:

assumes \exists size\_t off; 0<= off< length && array[off] == element;

ensures array<=\result<array+length && \*\result == element;

behavior noth:

assumes \forall size\_t off; 0<=off< length && array[off] != element;

ensures \result==NULL;

disjoint behaviors;

complete behaviors;

\*/

int\* search(int\* array,size\_t length,int element){

/\*@

loop invariant 0<=i<=length;

loop invariant \forall size\_t j; 0<=j<i ==> array[j]!=element;

loop assigns i;

loop variant length-i;

\*/

for(size\_t i=0;i<length;i++){

if(array[i]==element)

return &array[i];}

return NULL;

}

LOGIC FCN

Q)

#include<limits.h>

/\*@

logic integer ax\_b(integer a,integer x,integer b);

\*/

/\*@

lemma ax\_b\_pos:

\forall a,b,i,j;

a>0==>i<=j==>ax\_b(a,i,b)<ax\_b(a,j,b);

lemma ax\_b\_neg:

\forall a,b,i,j;

a<0==>i<=j==>ax\_b(a,i,b)>ax\_b(a,j,b);

lemma ax\_b\_eq:

\forall a,b,i,j;

a=0==>ax\_b(a,x,b)==ax\_b(a,j,b);

\*/

/\*@

requires INT\_MIN<=a\*x<INT\_MAX;

requires INT\_MIN<=ax\_b(a,x,b)<INT\_MAX;

assigns \nothing;

ensures \result==ax\_b(a,x,4);

\*/

int func(int a,int x){

return a\*x+4;

}

/\*@

requires INT\_MIN<=a\*x<INT\_MAX;

requires INT\_MIN<=a\*y<INT\_MAX;

requires a>0;

requires INT\_MIN<=ax\_b(a,x,4)<=INT\_MAX;

requires INT\_MIN<=ax\_b(a,y,4)<=INT\_MAX;

assigns \nothing;

\*/

void check(int a,int x,int y){

int fmin,fmax;

if(x>y){

fmin=func(a,x);

fmax=func(a,y);

}

else{

fmin=func(a,y);

fmax=func(a,x);

}

//@assert fmin<=fmax;

}

int main(){

int w=3;

int r=4;

int s=7;

int t=func(w,r);

check(w,r,s);

}

Q)  
#include<limits.h>

/\*@

logic integer ax\_b(integer a,integer x,integer b) = a\*x+b;

\*/

/\*@

requires 3\*x>=INT\_MIN;

requires INT\_MIN<=ax\_b(3,x,4)<INT\_MAX;

assigns \nothing;

ensures \result==ax\_b(3,x,4);

\*/

int fun(int x){

return 3\*x+4;

}

int main(){

int a=4;

int r=fun(a);

//@assert r==16;

}