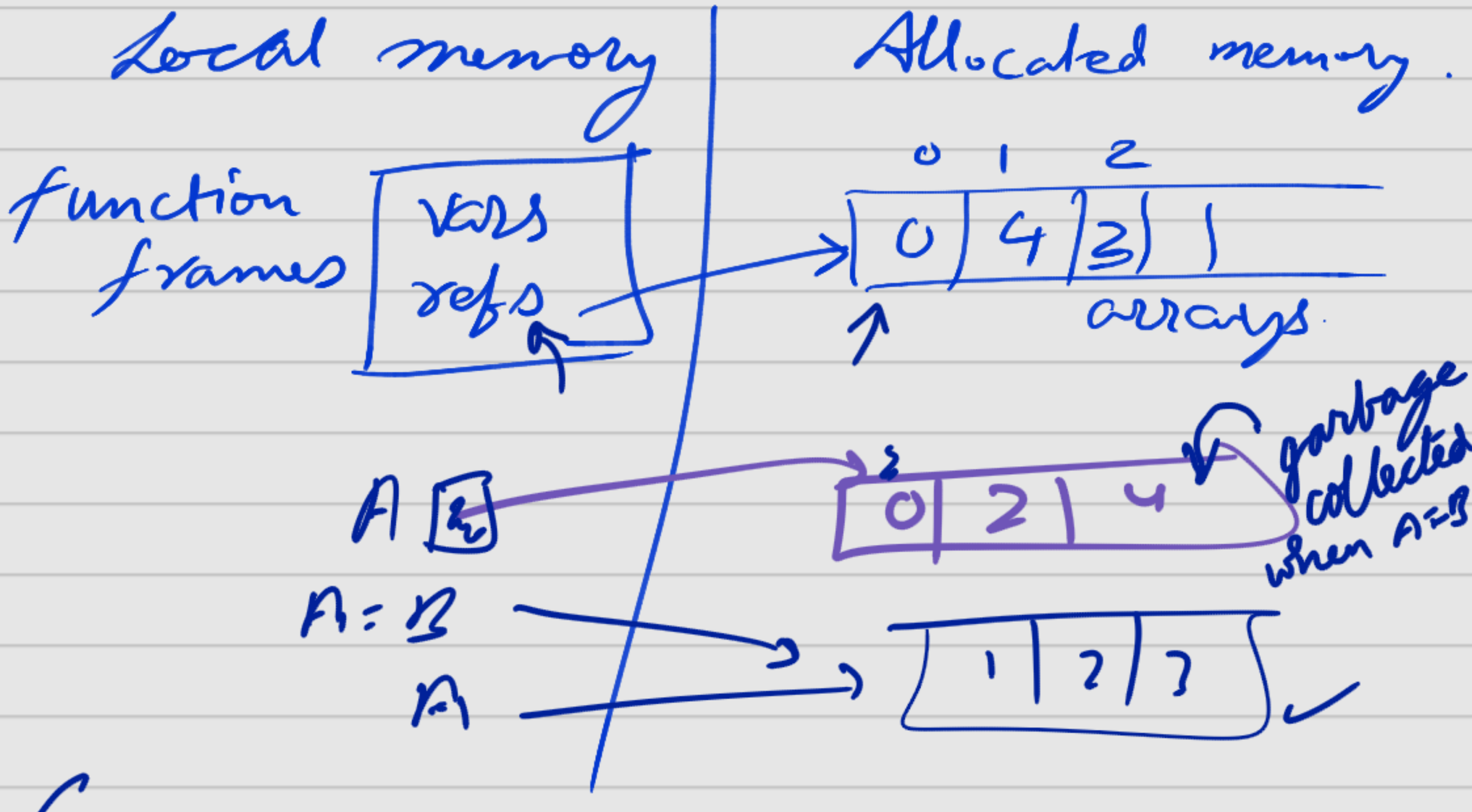
Recall that array vars are references (i.e. store the address) of arrays.

If one loses the address to an array then it becomes inaccessible and will be reclaimed or garbage collected.

Co memory model.

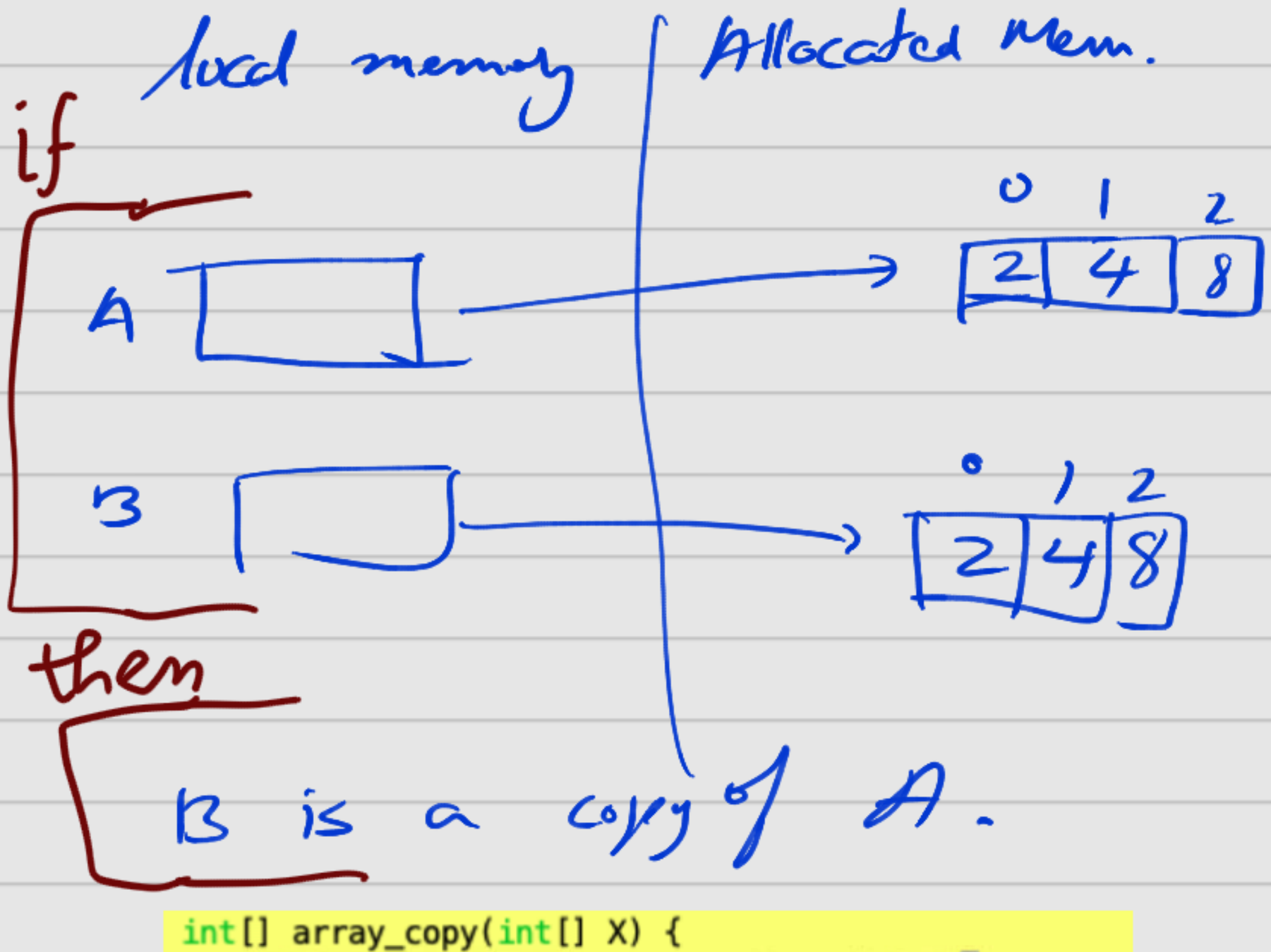



```
{
    int [] A = alloc-array(int, n);
    ⋮
    int [] B = A;
```

Programming with Arrays :-

Problem :- write a function which takes an int array and returns its copy.

Function name :- array_copy
Input :- array ....
Output :- array




```
int[] array_copy(int[] X) {
    // do something to create a copy of X.
}

int main() {
    int[] A;
    /* A multiline comment
    some code which fills entries in A comes here.
    */
    int[] B = array_copy(A);
    return 0;
}
```

Recall that:

int[] B = A; doesn't work due to array referencing (same as python?)

this just creates a new name (B) for the array pointed to by A.

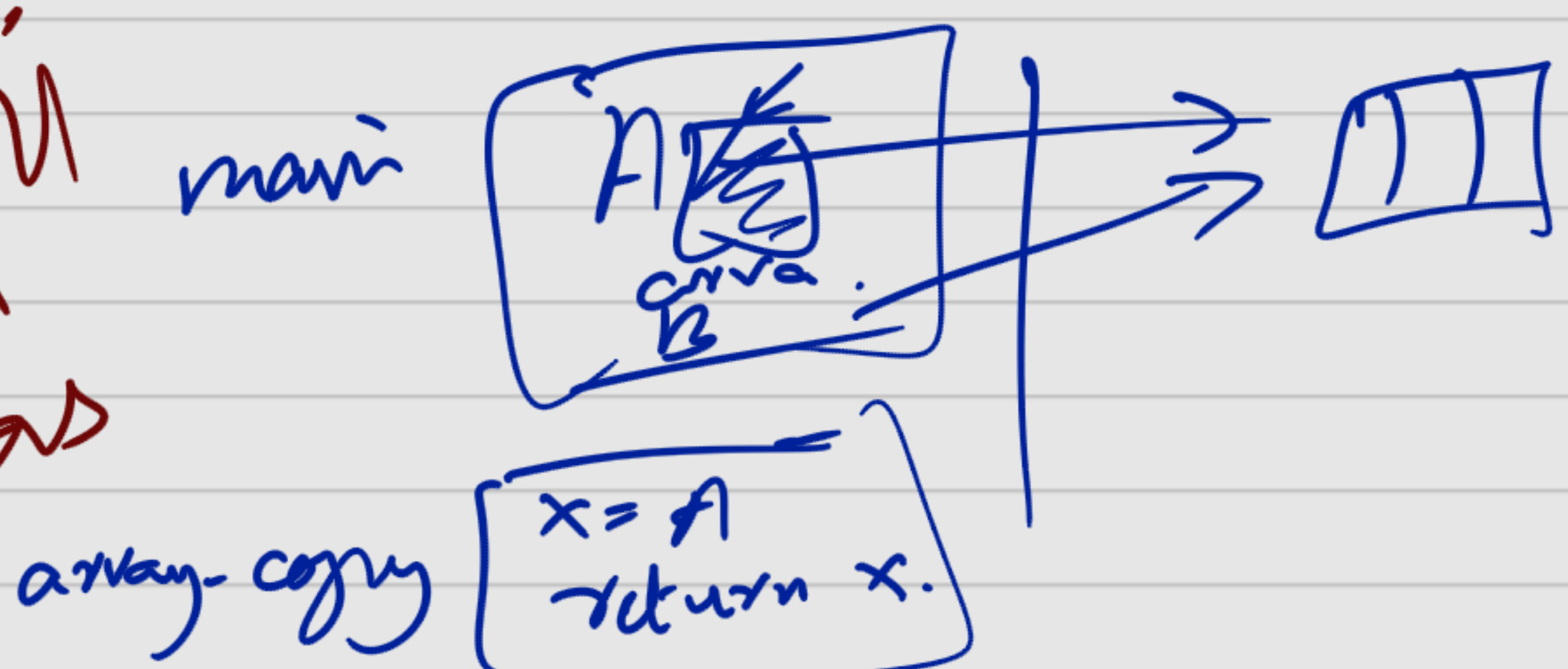
Does this work??

```
int[] array_copy(int[] X) {
    return X;
}

int main() {
    int[] A;
    /* A multiline comment
    some code which fills entries in A comes here.
    */
    int[] B = array_copy(A);
    return 0;
}
```

B = A

No, still an alias



We must allocate (or create) a new array.

Lets start writing it :-

```
int [] array_copy(int[] X){
→int[] C = alloc_array(int, ??);
    return C;
}
```

☆ Need to pass length of A also :-

```
int[] array_copy(int[] X, int n) {
→   int[] C = alloc_array(int, n); //put safety for this as a precondition.
    //do something more here.
    return C;
}

int main() {
    int[] A = alloc-array(int, 3);
    /* A multiline comment
    some code which fills entries in A.
    */
    int[] B = array_copy(A, 3); //lets say A had 3 entries
    return 0;
}
```

Preconditions for array-copy :-

```
//@ requires n >= 0;
```

Making sure that n is indeed the length of the passed array!

\length(X)

1 Contract only function -
2 Not available as a C0 function.
3 Returns length of X.

The post condition for \length is :-

\length(A)
//@ ensures \result >= 0;

Lets recall the other preconditions :-

last
class
alloc_array (Type, n)
//@ requires n >= 0;
//@ ensures \length(\result) == n;

A[i]
//@ requires 0 <= i && i < \length(A);

So lets use the \length contract function :-

```
int[] array_copy(int[] X, int n)
//@requires \length(X) == n;
{
    int[] C = alloc_array(int, n);
    C = X;
    return C;
}

int main() {
    int[] A;
    /* A multiline comment
    some code which fills entries in A.
    */
    int[] B = array_copy(A, 3); //lets say A had 3 entries
    return 0;
}
```

What goes wrong??

C is still an alias for X and forgets the newly created array from the alloc-array call.

Filling entries of X into C.

Final version:-

```
int[] array_copy(int[] X, int n)
//@requires \length(X) == n;
{
    int[] C = alloc_array(int, n);
    for(int i=0; i<n; i = i+1) {
        C[i] = X[i];
    }
    return C;
}

int main() {
    int[] A = alloc_array(int, 3);
    /* A multiline comment
    some code which fills entries in A.
    */
    int[] B = array_copy(A, 3); //lets say A had 3 entries
    return 0;
}
```

Analysing the array-copy fn:

① Safety :

```
int[] array_copy(int[] X, int n)
//@requires \length(X) == n;
{
    int[] C = alloc_array(int, n);
    for(int i=0; i<n; i = i+1) {
        C[i] = X[i];
    }
    return C;
}

int main() {
    int[] A = alloc_array(int, 3);
    /* A multiline comment
    some code which fills entries in A.
    */
    int[] B = array_copy(A, 3); //lets say A had 3 entries
    return 0;
}
```

L0 → add more

L1 → L3 → make sure these are safe calls.

Safety of alloc-array call in L1.

T.S. $n >= 0$:

① L0 : $n = \backslash length(X)$

② from postcondition on $\backslash length$ we know $\backslash length(X) >= 0$

① & ② $\Rightarrow n >= 0$ ∴

Q.E.D. □

Safety of A[i] (... and B[i]).
precondition for A[i]:
//@ requires 0 <= i
&&
i < \length(A);

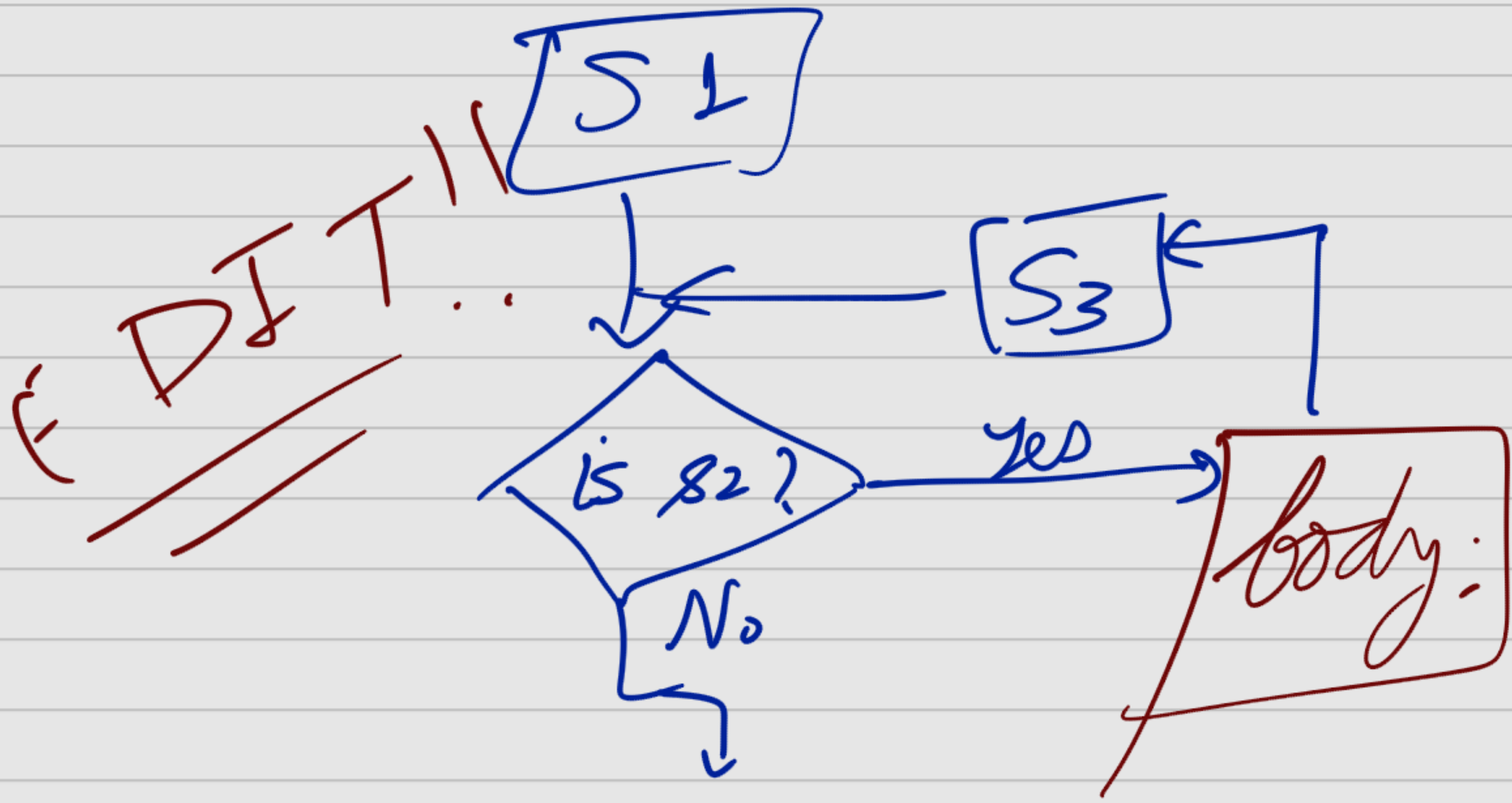
T.P.1) 0 <= i
T.P.2) i < \length(A);

```
int[] array_copy(int[] X, int n)
//@requires \length(X) == n;
{
    int[] C = alloc_array(int, n);
    for(int i=0; i<n; i = i+1) {
        C[i] = X[i];
    }
    return C;
}

int main() {
    int[] A = alloc_array(int, 3);
    /* A multiline comment
    some code which fills entries in A.
    */
    int[] B = array_copy(A, 3); //lets say A had 3 entries
    return 0;
}
```

L4 →

for (S1; S2; S3)
body.




```
L4 → for (int i=0; i<n; i=i+1)
     //@ loop-invariant 0<=i;
     {
       B[i]= A[i];
     }
```

By line L4 ; $i \geq 0$.

A[i]: i < \length(A).

```
int[] array_copy(int[] X, int n)
//@requires \length(X) == n;
{
    int[] C = alloc_array(int, n);
    for(int i=0; i<n; i = i+1) {
        C[i] = X[i];
    }
    return C;
}

int main() {
    int[] A = alloc_array(int, 3);
    /* A multiline comment
    some code which fills entries in A.
    */
    int[] B = array_copy(A, 3); //lets say A had 3 entries
    return 0;
}
```

L0 L1 L4 L3

① L0: n == \length(A)

② L4: i < n

① + ② ⇒ i < n == \length(A)

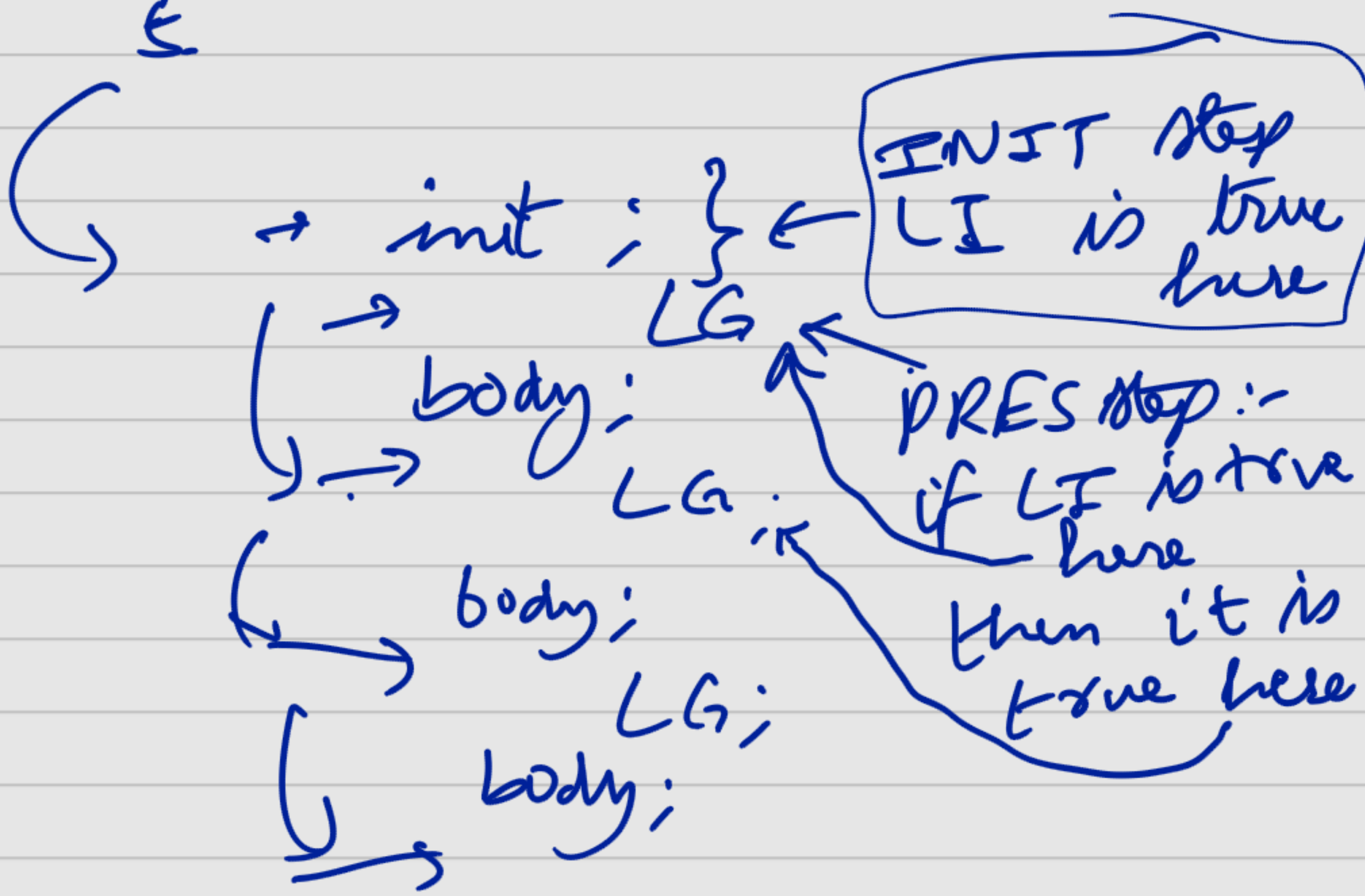
L4
→ for (int i=0; i<n; i=i+1)
//@ loop-invariant 0<=i;
{
B[i]= A[i];
}

Recall how to prove loop invariants.

INIT step: before loop is entered L.I. is true.

PRES step:- assume L.I. true before execution of loop body then prove that L.I. is true after execution of loop body.

init:
while (LG) {
body;
}



// @ loop_ invariant $0 <= i$;

INIT:

T.S. $0 <= i$ initially.

① L4 : $i = 0$

$0 \leq i$

$\therefore 0 \leq 0$

PRES step :-

i : before loop is executed once

i' : after loop is executed once

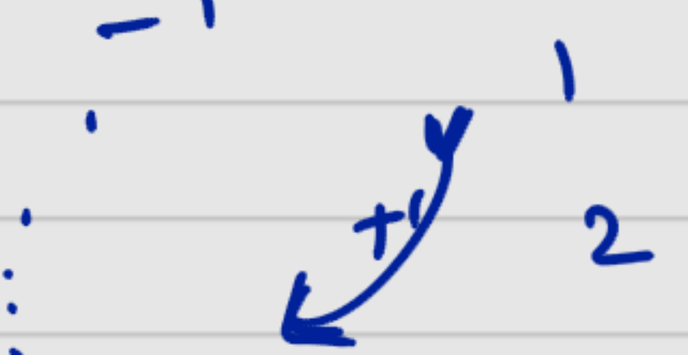
L4 : $i' = i + 1$

$0 \leq i$ by assumption.
LI. is true before loop execution.

$0 \leq i + 1$

↓

→ this is true only if $i \neq$ int_max() ∵ of two's complement



$i \neq$ INT_MAX) : as $i < n$ by L4

$\Rightarrow$ $0 \leq i$ ✓

```
int[] array_copy(int[] X, int n)
//@requires \length(X) == n;
{
    int[] C = alloc_array(int, n);
    for(int i=0; i<n; i = i+1) {
        C[i] = X[i];
    }
    return C;
}

int main() {
    int[] A = alloc_array(int, 3);
    /* A multiline comment
    some code which fills entries in A.
    */
    int[] B = array_copy(A, 3); //lets say A had 3 entries
    return 0;
}
```

int [] D = array-copy(B, 3);

the specification of array-copy is incomplete.

```
//@ensures n==\length(\result);
```

\length(D) = 3 ✓.

put this as a postcondition in array-copy fund.

Recall correctness :- (proving post condition)
preconditions imply postconditions :-

if $n == \backslash length(X)$;

$\Rightarrow$ $n == \backslash length(\backslash result)$;

```
int[] array_copy(int[] X, int n)
//@requires \length(X) == n;
{
    int[] C = alloc_array(int, n);
    for(int i=0; i<n; i = i+1) {
        C[i] = X[i];
    }
    return C;
}

int main() {
    int[] A = alloc_array(int, 3);
    /* A multiline comment
    some code which fills entries in A.
    */
    int[] B = array_copy(A, 3); //lets say A had 3 entries
    return 0;
}
```

L0 L1 L4 L3

① $n = \backslash length(X)$ by L0

② $\backslash length(C) == n$ by L1

③ $\backslash result = C$ by L3

substitute ③ in ②

$\therefore \backslash length(\backslash result) == n$

← end of proof → QED □

How do we know
C gets a copy of x?

Prove the following
loop invariants:

1. $0 \leq i \leq n$
2. $\forall j$ st $0 \leq j < i$
   we have $C[j] = X[j]$

before
Ex for
assignment 11