Loop Invariants

Loops are harder to prove
- correctness at every step
  - loop termination

Loop Invariant:
A condition which should hold true inside a loop
Finding Max Pos

```c
main () {
    int posmax;
    int A[10];
    srand(time(NULL));
    for (int i=0; i<10; i++) A[i] = rand()%100;
    for (int i=0; i<10; i++) {
        if (A[i]>A[posmax]) posmax=i;
    }
    for (int i=0; i<10; i++) cout << A[i] << " ";
    cout << endl << A[posmax] << endl;
}
```

- There is a mistake in this program which goes undetected
- The program is to compute the position in the array, where the greatest element is located
- Everytime you run the program, a random array is generated
main () {

int posmax;
int A[10];

srand(time(NULL));
for (int i=0; i<10; i++) A[i] = rand()%100;

for (int i=0; i<10; i++) {
    assert(i>=0);
    assert(i<10);
    assert(posmax>=0);
    assert(posmax<10);
    
    if (A[i]>A[posmax]) posmax=i;
}
assert(posmax>=0);
assert(posmax<10);

for (int i=0; i<10; i++) cout << A[i] << " ";
cout << endl << A[posmax] << endl;
}
```cpp
main () {
    int posmax=0;
    int A[10];

    srand(time(NULL));
    for (int i=0; i<10; i++) A[i] = rand()%100;

    for (int i=0; i<10; i++) {
        assert(i>=0);
        assert(i<10);
        assert(posmax>=0);
        assert(posmax<10);
        if (A[i]>A[posmax]) posmax=i;
    }
    assert(posmax>=0);
    assert(posmax<10);

    for (int i=0; i<10; i++) cout << A[i] << " ";
    cout << endl << A[posmax] << endl;
}
```

- Initialization of variable posmax was missing.
- It used a garbage value
- If the garbage was within range, it would not have been detected by the loop invariant
- However, it is okay to start with any valid value as max, which is updated after every iteration
- So we donot need the loop precondition for entry.
main () {

int posmax=0;
int A[10];

srand(time(NULL));
for (int i=0; i<10; i++) A[i] = rand()%100;
assert(posmax>=0);
assert(posmax<10);
for (int i=0; i<10; i++) {
    assert(i>=0);
    assert(i<10);
    assert(posmax>=0);
    assert(posmax<10);
    if (A[i]>A[posmax]) posmax=i;
}
assert(posmax>=0);
assert(posmax<10);

for (int i=0; i<10; i++) cout << A[i] << " ";
cout << endl << A[posmax] << endl;
}

- To cover the initialization of the variables accessed inside the loop, we may use a precondition for the entry into the loop.
- The loop preconditions can be eliminated if they are redundant with the first execution of the loop invariant.
  - which is the case for this program.
main () {

int posmax=0;
int A[10];

srand(time(NULL));
for (int i=0; i<10; i++) A[i] = rand()%100;
assert(posmax>=0);
assert(posmax<10);
for (int i=0; i<10; i++) {
    assert(i>=0);
    assert(i<10);
    assert(posmax>=0);
    assert(posmax<10);
    if (A[i]>A[posmax]) posmax=i;
}
assert(posmax>=0);
assert(posmax<10);
for (int i=0; i<10; i++) cout << A[i] << " ";
cout << endl << A[posmax] << endl;
}

- Observe the loop postcondition, which is outside the loop
- Just like redundant loop preconditions, if a loop postcondition outside the loop is semantically (i.e. the meaning of it) redundant with the loop invariant, it can be removed
- In this case, we cannot remove it since there is a state change after the loop invariant. The state change changes the variables accessed in the assertion
Correctness of Loop Action not checked

Adding another meaningful Loop invariant

- Note we removed the redundant loop precondition
- The loop action is incorrect
- But all assertions are satisfied
- Something is missing!

```c
main () {

    int posmax=0;
    int A[10];

    srand(time(NULL));
    for (int i=0; i<10; i++) A[i] = rand()%100;

    for (int i=0; i<10; i++) {
        assert(i>=0);
        assert(i<10);
        assert(posmax>=0);
        assert(posmax<10);

        if (A[i]<A[posmax]) posmax=i;
    }
    assert(posmax>=0);
    assert(posmax<10);

    for (int i=0; i<10; i++) cout << A[i] << " ";
    cout << endl << A[posmax] << endl;
}
```
Adding a Loop Invariant for checking the correctness of loop action

main () {

    int posmax=0;
    int A[10];

    srand(time(NULL));
    for (int i=0; i<10; i++) A[i] = rand()%100;

    for (int i=0; i<10; i++) {
        assert(i>=0);
        assert(i<10);
        assert(posmax>=0);
        assert(posmax<10);

        if (A[i]<A[posmax]) posmax=i;
        assert(partdone(A,i,A[posmax]));
    }
    assert(posmax>=0);
    assert(posmax<10);

    for (int i=0; i<10; i++) cout << A[i] << " ";
    cout << endl << A[posmax] << endl;
}
Making loop invariant stronger

- We move the loop postcondition into the loop, making it loop invariant

- It would check in every iteration for the correct value of i

- But could we move it before assertion partdone?

```c
main () {

    int posmax=0;
    int A[10];

    srand(time(NULL));
    for (int i=0; i<10; i++) A[i] = rand()%100;

    for (int i=0; i<10; i++) {
        assert(i>=0);
        assert(i<10);
        assert(posmax>=0);
        assert(posmax<10);

        if (A[i]>A[posmax]) posmax=i;
        assert(partdone(A,i,A[posmax]));
        assert( (posmax>=0)&&(posmax<10));
    }
    assert(posmax>=0);
    assert(posmax<10);

    for (int i=0; i<10; i++) cout << A[i] << " ";
    cout << endl << A[posmax] << endl;
```
Our Solution

- We move the loop postcondition into the loop, making it loop invariant
- It would check in every iteration for the correct value of i
- But could we move it before assertion `partdone`?
Compaction of assertion: combing them into *conjunctions*

- combining the assertions
- But we lose on error reporting, as the C assert macro does not report exactly which component of the assertion failed

```c
main () {

    int posmax=0;
    int A[10];

    srand(time(NULL));
    for (int i=0; i<10; i++) A[i] = rand()%100;

    for (int i=0; i<10; i++) {
        assert ((i>=0) && (i<10));
        assert ((posmax>=0) && (posmax<10));

        if (A[i]>A[posmax]) posmax=i;

        asseer( (posmax>=0)&&(posmax<10));
        assert(partdone(A,i,A[posmax]));
    }
    for (int i=0; i<10; i++) cout << A[i] << " ";
    cout << endl << A[posmax] << endl;
}
```
Improving the efficiency of assertion checking

- we know in this case that the assertion in the end will also be checked in the beginning of the next iteration
- except for the entry
- So we move it out as loop precondition

```cpp
main () {

int posmax=0;
int A[10];

srand(time(NULL));
for (int i=0; i<10; i++) A[i] = rand()%100;
assert ((posmax>=0) && (posmax<10));
for (int i=0; i<10; i++) {
    assert ((i>=0) && (i<10));
    assert ((posmax>=0) && (posmax<10));
    if (A[i]>A[posmax]) posmax=i;
    asseer( (posmax>=0)&&(posmax<10));
    assert(partdone(A,i,A[posmax]));
}
for (int i=0; i<10; i++) cout << A[i] << " ";
cout << endl << A[posmax] << endl;
}
```