Inspection and Procedural Abstraction

Course theme: problem decomposition

- Achieve control of a complex system by
  1. Dividing it into relatively independent parts.
  2. Documenting the interdependencies that remain.

- Steps 1 and 2 are both important.

- The result of a well-designed and carefully documented decomposition is a system that is easier to understand, inspect, test, and modify.

Procedural abstraction

- *Abstraction*: intentionally ignoring certain aspects of a problem to simplify analysis and to focus attention on the remaining aspects.

- In *procedural abstraction* the procedure implementation is ignored to focus attention on the service provided by the procedure.

- Successful procedural abstraction requires a precise and complete specification of the service offered by the procedure.

Procedural abstraction in inspection

- Suppose that you are given specifications and implementations for the C functions `F` and `G`, and that `F` calls `G`.

- To inspect `G`: show that `G`'s implementation behaves as required by its specification.

- To inspect `F`: show that `F`'s implementation behaves as required by its specification. When reasoning about the call to `G` in `F`'s implementation, refer to `G`'s specification and *not* its implementation.

Example

- Use procedural abstraction to inspect the `findLongestPlateau` and `removeLongestPlateau` functions shown below.

- First show that `findLongestPlateau` is correct or produce a list of the faults found.

- Then show that `removeLongestPlateau` is correct or produce a list of the faults found. When reasoning about the call to `findLongestPlateau`, refer to its specification *not* its implementation.

- Note: A *plateau* in a sequence of numbers is a subsequence of one or more consecutive numbers of the same value. In the sequence

```
S = <1,2,2,3,2,0,0,0>
```

there are many plateaus including

```
<2,2>
<3>
<0>
<0,0>
<0,0,0>
```

In S, the *longest plateau* is <0,0,0>. If a sequence has two or more "longest plateaus" then *longest plateau* refers to the leftmost one.

```
/* Assign to *pStart and *pLen the starting position and length
 * of the longest plateau in a[0..aLen-1].
 *
 * Assumed: a has at least aLen elements and aLen > 0
 */
void findLongestPlateau(int a[],int aLen,int* pStart,int *pLen)
{
        int tmpStart,tmpLen,i;

        *pStart = 0;
        *pLen = 1;
        tmpStart = 0;
        tmpLen = 1;
        for (i = 0; i < aLen-1; i++) {
                if (a[i] == a[i+1]) {
                        tmpLen++;
                } else {
                        if (tmpLen >= *pLen) {
                                *pStart = tmpStart;
                                *pLen = tmpLen;
                        }
                        tmpStart = i+1;
                        tmpLen = 1;
                }
        }
}

/* Remove P, the longest plateau in a, shifting left all the elements
 * to the right of P and decreasing *aLen appropriately.
 *
 * Assumed: a has at least *aLen elements and *aLen > 0
 */
void removeLongestPlateau(int a[],int *aLen)
{
        int i,pStart,pLen;

        findLongestPlateau(a,*aLen,&pStart,&pLen);
        for (i = 0; i < *aLen-pStart-pLen; i++) {
                a[i+pStart] = a[i+pStart+pLen];
        }
}

int main()
{
```

```
    int i,xLen,x[100];

    /* longest plateau at end */
    xLen = 4; x[0] = 1; x[1] = 2; x[2] = 3; x[3] = 3;
    removeLongestPlateau(x,&xLen);
    for (i = 0; i < xLen; i++)
        printf("%d\n",x[i]);
    printf("\n");

    /* two longest plateaus */
    xLen = 6; x[0] = 1; x[1] = 2; x[2] = 2; x[3] = 3; x[4] = 3; x[5] = 1;
    removeLongestPlateau(x,&xLen);
    for (i = 0; i < xLen; i++)
        printf("%d\n",x[i]);
}
```