

CSCI 135 Software Design and Analysis I

Homework 6

Solution

Saad Mneimneh
Visiting Professor
Hunter College of CUNY

Problem 1

Do labs 9 (input and output strings) and 11 (two dimensional arrays).

Problem 2: Recursion with Palindromes

A palindrome is a string that spells the same forwards and backwards, e.g. “radar”. Given a string s , we would like to determine if $s[i] \dots s[j]$ is a palindrome.

Let us first find a recursive definition of palindrome. A string $s[i] \dots s[j]$ is a palindrome if $s[i+1] \dots s[j-1]$ is a palindrome and $s[i] = s[j]$. Therefore

$$palindrome(s, i, j) = \begin{cases} (s[i] == s[j]) \text{ and } palindrome(s, i+1, j-1) & j > i \\ true & j \leq i \end{cases}$$

(a) Write a recursive function for the above definition to determine whether a string is a palindrome or not.

ANSWER:

```
bool palindrome(const char * s, int i, int j) {
    if (i < j)
        return (s[i] == s[j] && palindrome(s, i+1, j-1));
    else
        return true;
}
```

(b) The definition above is not tail recursive because after the function call to `palindrome` returns, the result must be ANDed with $s[i] == s[j]$. Using an extra `if`, transform the recursion into a tail recursion.

ANSWER:

```
bool palindrome(const char * s, int i, int j) {
    if (i<j) {
        if (s[i]==s[j])
            return palindrome(s, i+1, j-1);
        else
            return false;
    }
    else
        return true;
}
```

(c) Transform the tail recursive implementation into an iterative one.

```
bool palindrome(const char * s, int i, int j) {
    while (i<j) { //replace if with while
        if (s[i]==s[j]) {
            i++; //replace recursive
            j--; //call with updates
        }
        else
            return false;
    }
    //remove else
    return true;
}
```

We can simplify this further by noting that every return statement (2 of them) returns $(j \leq i)$.

```
bool palindrome(const char * s, int i, int j) {
    while (i<j) { //replace if with while
        if (s[i]==s[j]) {
            i++; //replace recursive
            j--; //call with updates
        }
        else
            return j<=i;
    }
    //remove else
    return j<=i;
}
```

Now we can put the if condition within the while, and join both returns in one.

```
bool palindrome(const char * s, int i, int j) {
    while (i<j && s[i]==s[j]) {
        i++;
        j--;
    }
    return j<=i;
}
```

(d) With the help of the function `palindrome` in (c), write a function (call it `largestPalindrome`) to find the largest palindrome contained in a string, using 3 nested loops (including the loop in function `palindrome`).

ANSWER:

```
void largestPalindrome(const char * s) {
    int largest=0;
    int index=0;
    int l=strlen(s);
    for (int i=0; i<l; i++)
        for (int j=i; j<l; j++)
            if (palindrome(s,i,j) && j-i+1>largest) {
                largest=j-i+1;
                index=i;
            }
    for (int i=index; i<index+largest; i++)
        cout<<s[i];
    cout<<'\n';
}
```

(e) The `palindrome` function in (c) works inward by checking extremities. Write another function `palindrome2` that finds a palindrome in a string by working outward given the center of the palindrome. Make it return the length of the largest such palindrome. Use it to solve (d) using only 2 nested loops (including the loop in the function `palindrome2`).

ANSWER: Given a string with length l , we either consider the i^{th} character as a center, for $i = 0..l - 1$, or the empty string between the i^{th} and $(i + 1)^{st}$ character as a center, for $i = 0..l - 2$. The function returns the length of the largest palindrome with that center, i.e. 0 when no palindrome (the length can be used as a boolean).

```
int palindrome2(const char * s, int i, bool between) {
    int l=strlen(s);
    int length;
    int j;
    if (between) {
        j=i+1; length=0;
    }
    else {
        j=i+1; i=i-1; length=1;
    }
    while (s[i]==s[j] && i>-1 && j<l) {
        i--;
        j++;
        length=length+2;
    }
    return length;
}
```

```
void largestPalindrome(const char * s) {
    int largest=0;
    int index=0;
    bool between;
    int l=strlen(s);
    for (int i=0; i<l; i++) {
        if (int len=palindrome2(s,i,0)>largest) {
            largest=len;
            index=i;
            between=0;
        }
        if (int len=palindrome(s,i,1)>largest) {
            largest=len;
            index=i;
            between=1;
        }
    }
    if (between) {
        //print the corresponding palindrome
    }
    else {
        //print the corresponding palidrome
    }
}
```