```
bool is_palindrome(const string& s) {
    string s_cp;
    for (int i = s.size()-1; i>=0; --i) s_cp.push_back(s[i]);
    if (s==s_cp) return true;
    return false;
}
bool is_palindrome(const char s[], int n) {
    char* s_cp = new char[n+1];
    s_cp[n] = 0;
    for (int i = 0; i<n; ++i) s_cp[n-1-i] = s[i];
    if (strcmp(s,s_cp)) {
        delete[] s_cp;
        return false;
    }
    delete[] s_cp;
    return true;
}
bool is_palindrome(const char* first, const char* last) {
    char* s_cp = new char[last-first+2];
    char* s_cp_ptr = s_cp;
    while (last>=first) {
        *s_cp_ptr = *last;
        ++s_cp_ptr;
        --last;
    }
    *s_cp_ptr = 0;
    if (strcmp(first,s_cp)) {
        delete[] s_cp;
        return false;
    }
    delete[] s_cp;
    return true;
}
istream& read_word(istream& is, char* buffer, int max) {
    is.width(max);
    is >> buffer;
    return is;
}
int main()
{
    string s;
    while (cin>>s && s!="quit") {
        cout << s << " is";
        if (!is_palindrome(s)) cout << " not";
        cout << " a palindrome\n";
    }
    const int max = 128;
    char s2[max];
    while (read_word(cin,s2,max) && !strcmp(s2,"quit")) {
        cout << s2 <<  " is";
        if (!is_palindrome(s2,strlen(s2))) cout << " not";
        cout << " a palindrome\n";
    }
    while (read_word(cin,s2,max) && !strcmp(s2,"quit")) {
        cout << s2 << " is";
        if (!is_palindrome(&s2[0],&s2[strlen(s2)-1])) cout << " not";
        cout << " a palindrome\n";
    }
}
```

1. Compare and contrast the three `while` loops in `main()`.
   ANS: All three while loops call a `bool` function named `is_palindrome()`, but each is overloaded with a different

parameter list–which function is executed depends on the parameter list.

The first `while` loop reads in a `string s`. If the string read is not "`quit`" it calls the function defined on lines 1-6 which uses a `for` toop to make a copy of the string in reverse order and then the comparison operator `==` is used to see if the reverse string is the same as the string passed in, if so, it returns `true`, else it returns `false.`

The second `while` loop is conditioned upon the `AND` of two `bool`s, the first of which is the output of the `read_word()` function. If the `read_word()` function successfully reads a C-string into `s2`, then the input stream reference (`istream&`) returned will be interpretted in this context as "`true`"–leading the compiler to check the output of `strcmp(s2,"quit"` which compares the C-string s2 to the C-string `quit` and returns `false` if they are equal.

So unless the input stream is corrupted somehow, or the user enters "`quit`", this second `while` loop calls the `is_palindrome()` function on lines 7-17 that takes a C-string and an `int`. It copies the C-string `s2` into another C-string allocated on the heap, only backwards and uses `strcmp()` to see if the reverse is the same as the C-string passed in. If so, it returns `true`, or returns `false` otherwise.

The third `while` loop uses the same input strategy but calls the `is_palindrome()` function with the first parameter as a C-string by the address of the first character of `s2`, and the second parameter as a C-string with the last non-`NUL` character of `s2`. The idea here is to read characters forward from the beginning into one C-string and read characters backwards from the other C-string until we get to the middle, and then compare these two strings–if they're the same we have a palindrome! To do this, we have to preserve the values, so we allocate enough memory on the heap for the whole word + the `NUL` and create a pointer to `char, s_cp` to point to that and make a copy of it, `s_cp_pt`: yes, this is necessary...however awkward! Then we go about comparing the characters in the string pointed to by this copy and increment it while decrementing `last` and copying `*last` into `*s_cp_ptr` as we go. Then we put the `NUL` character at the end of the C-string pointed to by `s_cp` and compare `s_cp` with `first`. If they're the same we delete the memory we allocated and return `true`, otherside we delete the memory allocated and return `false.`