3.7 Break and Continue It is sometimes convenient to be able to exit from a loop other than by testing at the top or bottom. The break statement provides an early exit from for, while, and do, just as from switch. A break causes the innermost enclosing loop or switch to be exited immediately. The following function, trim, removes trailing blanks, tabs and newlines from the end of a string, using a break to exit from a loop when the rightmost non-blank, non-tab, non-newline is found. /* trim: remove trailing blanks, tabs, newlines */ int trim(char s[]) { int n; for (n = strlen(s)-1; n >= 0; n--) if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n') break; s[n+1] = '\0'; return n; } strlen returns the length of the string. The for loop starts at the end and scans backwards looking for the first character that is not a blank or tab or newline. The loop is broken when one is found, or when n becomes negative (that is, when the entire string has been scanned). You should verify that this is correct behavior even when the string is empty or contains only white space characters. The continue statement is related to break, but less often used; it causes the next iteration of the enclosing for, while, or do loop to begin. In the while and do, this means that the test part is executed immediately; in the for, control passes to the increment step. The continue statement applies only to loops, not to switch. A continue inside a switch inside a loop causes the next loop iteration. As an example, this fragment processes only the non-negative elements in the array a; negative values are skipped. for (i = 0; i < n; i++) if (a[i] < 0) /* skip negative elements */ continue; ... /* do positive elements */ The continue statement is often used when the part of the loop that follows is complicated, so that reversing a test and indenting another level would nest the program too deeply.