

K&R 1.1

```
#include <stdio.h>
/* print Fahrenheit-Celsius table
   for fahr = 0, 20, ..., 300 */
main()
{
    int fahr, celsius;
    int lower, upper, step;
    lower = 0;          /* lower limit of temperature scale */
    upper = 300;         /* upper limit */
    step = 20;           /* step size */
    fahr = lower;
    while (fahr <= upper) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + step;
    }
}
```

K&R 1.2

```
#include <stdio.h>

/* print Fahrenheit-Celsius table
   for fahr = 0, 20, ..., 300; floating-point version */
main()
{
    float fahr, celsius;
    float lower, upper, step;

    lower = 0;          /* lower limit of temperatuire scale */
    upper = 300;        /* upper limit */
    step = 20;          /* step size */

    fahr = lower;
    while (fahr <= upper) {
        celsius = (5.0/9.0) * (fahr-32.0);
        printf("%3.0f %6.1f\n", fahr, celsius);
        fahr = fahr + step;
    }
}
```

K&R 1.3

```
#include <stdio.h>
/* copy input to output; 1st version */
main()
{
    int c;

    c = getchar();
    while (c != EOF) {
        putchar(c);
        c = getchar();
    }
}
```

```
#include <stdio.h>
/* copy input to output; 2nd version */
main()
{
    int c;

    while ((c = getchar()) != EOF)
        putchar(c);
}
```

K&R 1.4

```
#include <stdio.h>
#define IN    1 /* inside a word */
#define OUT   0 /* outside a word */

/* count lines, words, and characters in input */
main()
{
    int c, nl, nw, nc, state;

    state = OUT;
    nl = nw = nc = 0;
    while ((c = getchar()) != EOF) {
        ++nc;
        if (c == '\n')
            ++nl;
        if (c == ' ' || c == '\n' || c == '\t')
            state = OUT;
        else if (state == OUT) {
            state = IN;
            ++nw;
        }
    }
    printf("%d %d %d\n", nl, nw, nc);
}
```

K&R 1.5

```
#include <stdio.h>
/* count digits, white space, others */
main()
{
    int c, i, nwhite, nother;
    int ndigit[10];

    nwhite = nother = 0;
    for (i = 0; i < 10; ++i)
        ndigit[i] = 0;
    while ((c = getchar()) != EOF)
        if (c >= '0' && c <= '9')
            ++ndigit[c-'0'];
        else if (c == ' ' || c == '\n' || c == '\t')
            ++nwhite;
        else
            ++nother;
    printf("digits =");
    for (i = 0; i < 10; ++i)
        printf(" %d", ndigit[i]);
    printf(", white space = %d, other = %d\n",
           nwhite, nother);
}
```

K&R 1.6-1

```
#include <stdio.h>
#define MAXLINE 1000 /* maximum input line length */

int getline(char line[], int maxline);
void copy(char to[], char from[]);

/* print the longest input line */
main()
{
    int len;                  /* current line length */
    int max;                  /* maximum length seen so far */
    char line[MAXLINE];      /* current input line */
    char longest[MAXLINE];   /* longest line saved here */

    max = 0;
    while ((len = getline(line, MAXLINE)) > 0)
        if (len > max) {
            max = len;
            copy(longest, line);
        }
    if (max > 0) /* there was a line */
        printf("%s", longest);
    return 0;
}
```

K&R 1.6-2

```
/* getline:  read a line into s, return length  */
int getline(char s[], int lim)
{
    int c, i;

    for (i=0; i < lim-1 && (c=getchar())!=EOF && c!='\n'; ++i)
        s[i] = c;
    if (c == '\n') {
        s[i] = c;
        ++i;
    }
    s[i] = '\0';
    return i;
}

/* copy:  copy 'from' into 'to'; assume to is big enough */
void copy(char to[], char from[])
{
    int i;

    i = 0;
    while ((to[i] = from[i]) != '\0')
        ++i;
}
```

K&R 2.1

```
/* atoi: convert s to integer */
int atoi(char s[])
{
    int i, n;

    n = 0;
    for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
        n = 10 * n + (s[i] - '0');
    return n;
}

/* lower: convert c to lower case; ASCII only */
int lower(int c)
{
    if (c >= 'A' && c <= 'Z')
        return c + 'a' - 'A';
    else
        return c;
}
```

K&R 2.2

```
/* strcat: concatenate t to end of s; s must be big enough */
void strcat(char s[], char t[])
{
    int i, j;

    i = j = 0;
    while (s[i] != '\0') /* find end of s */
        i++;
    while ((s[i++] = t[j++]) != '\0') /* copy t */
        ;
}
```

```
/* bitcount: count 1 bits in x */
int bitcount(unsigned x)
{
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```

K&R 3.1

```
/* binsearch:  find x in v[0] <= v[1] <= ... <= v[n-1] */  
int binsearch(int x, int v[], int n)  
{  
    int low, high, mid;  
  
    low = 0;  
    high = n - 1;  
    while (low <= high) {  
        mid = (low+high)/2;  
        if (x < v[mid])  
            high = mid + 1;  
        else if (x > v[mid])  
            low = mid + 1;  
        else /* found match */  
            return mid;  
    }  
    return -1; /* no match */  
}
```

```
#include <stdio.h>
main() /* count digits, white space, others */
{
    int c, i, nwhite, nother, ndigit[10];

    nwhite = nother = 0;
    for (i = 0; i < 10; i++)
        ndigit[i] = 0;
    while ((c = getchar()) != EOF) {
        switch (c) {
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
            ndigit[c-'0']++;
            break;
        case ' ':
        case '\n':
        case '\t':
            nwhite++;
            break;
        default:
            nother++;
            break;
        }
    }
    printf("digits =");
    for (i = 0; i < 10; i++)
        printf(" %d", ndigit[i]);
    printf(", white space = %d, other = %d\n",
           nwhite, nother);
    return 0;
}
```

K&R 3.2

K&R 3.3

```
#include <ctype.h>

/* atoi: convert s to integer; version 2 */
int atoi(char s[])
{
    int i, n, sign;

    for (i = 0; isspace(s[i]); i++) /* skip white space */
        ;
    sign = (s[i] == '-') ? -1 : 1;
    if (s[i] == '+' || s[i] == '-') /* skip sign */
        i++;
    for (n = 0; isdigit(s[i]); i++)
        n = 10 * n + (s[i] - '0');
    return sign * n;
}
```

K&R 3.4

```
/* shellsort:  sort v[0]...v[n-1] into increasing order */
void shellsort(int v[], int n)
{
    int gap, i, j, temp;

    for (gap = n/2; gap > 0; gap /= 2)
        for (i = gap; i < n; i++)
            for (j=i-gap; j>=0 && v[j]>v[j+gap]; j-=gap) {
                temp = v[j];
                v[j] = v[j+gap];
                v[j+gap] = temp;
            }
}
```

K&R 3.5

```
#include <string.h>

/* reverse:  reverse string s in place */
void reverse(char s[])
{
    int c, i, j;

    for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

K&R 3.6

```
/* itoa: convert n to characters in s */
void itoa(int n, char s[])
{
    int i, sign;

    if ((sign = n) < 0) /* record sign */
        n = -n;           /* make n positive */
    i = 0;
    do {             /* generate digits in reverse order */
        s[i++] = n % 10 + '0'; /* get next digit */
    } while ((n /= 10) > 0); /* delete it */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}
```

K&R 3.7

```
/* 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;
}
```

K&R 4.1-1

```
#include <stdio.h>
#define MAXLINE 1000 /* maximum input line length */

int getline(char line[], int max)
int strindex(char source[], char searchfor[]);

char pattern[] = "ould"; /* pattern to search for */

/* find all lines matching pattern */
main()
{
    char line[MAXLINE];
    int found = 0;

    while (getline(line, MAXLINE) > 0)
        if (strindex(line, pattern) >= 0) {
            printf("%s", line);
            found++;
        }
    return found;
}
```

```
/* getline: get line into s, return length */
int getline(char s[], int lim)
{
    int c, i;

    i = 0;
    while (--lim > 0 && (c=getchar()) != EOF && c != '\n')
        s[i++] = c;
    if (c == '\n')
        s[i++] = c;
    s[i] = '\0';
    return i;
}

/* strindex: return index of t in s, -1 if none */
int strindex(char s[], char t[])
{
    int i, j, k;
    for (i = 0; s[i] != '\0'; i++) {
        for (j=i, k=0; t[k]!='\0' && s[j]==t[k]; j++, k++)
            ;
        if (k > 0 && t[k] == '\0')
            return i;
    }
    return -1;
}
```

K&R 4.2

```
#include <ctype.h>

/* atof: convert string s to double */
double atof(char s[])
{
    double val, power;
    int i, sign;

    for (i = 0; isspace(s[i]); i++) /* skip white space */
        ;
    sign = (s[i] == '-') ? -1 : 1;
    if (s[i] == '+' || s[i] == '-')
        i++;
    for (val = 0.0; isdigit(s[i]); i++)
        val = 10.0 * val + (s[i] - '0');
    if (s[i] == '.')
        i++;
    for (power = 1.0; isdigit(s[i]); i++) {
        val = 10.0 * val + (s[i] - '0');
        power *= 10;
    }
    return sign * val / power;
}
```

K&R 4.3-1

```
#include <stdio.h>
#include <stdlib.h> /* for atof() */

#define MAXOP    100 /* max size of operand or operator */
#define NUMBER   '0' /* signal that a number was found */

int getop(char []);
void push(double);
double pop(void);

/* reverse Polish calculator */
main()
{
    int type;
    double op2;
    char s[MAXOP];

    while ((type = getop(s)) != EOF) {
        switch (type) {
        case NUMBER:
            push(atof(s));
            break;
        case '+':
            push(pop() + pop());
            break;
```

K&R 4.3-2

```
case '*':
    push(pop() * pop());
    break;
case '-':
    op2 = pop();
    push(pop() - op2);
    break;
case '/':
    op2 = pop();
    if (op2 != 0.0)
        push(pop() / op2);
    else
        printf("error: zero divisor\n");
    break;
case '\n':
    printf("\t%.8g\n", pop());
    break;
default:
    printf("error: unknown command %s\n", s);
    break;
}
return 0;
}
```

K&R 4.3-3

```
#define MAXVAL 100 /* maximum depth of val stack */

int sp = 0;           /* next free stack position */
double val[MAXVAL]; /* value stack */

/* push: push f onto value stack */
void push(double f)
{
    if (sp < MAXVAL)
        val[sp++] = f;
    else
        printf("error: stack full, can't push %g\n", f);
}

/* pop: pop and return top value from stack */
double pop(void)
{
    if (sp > 0)
        return val[--sp];
    else {
        printf("error: stack empty\n");
        return 0.0;
    }
}
```

K&R 4.3-4

```
#include <ctype.h>

int getch(void);
void ungetch(int);

/* getop:  get next character or numeric operand */
int getop(char s[])
{
    int i, c;

    while ((s[0] = c = getch()) == ' ' || c == '\t')
        ;
    s[1] = '\0';
    if (!isdigit(c) && c != '.')
        return c;          /* not a number */
    i = 0;
    if (isdigit(c))      /* collect integer part */
        while (isdigit(s[++i] = c = getch()))
            ;
    if (c == '.')        /* collect fraction part */
        while (isdigit(s[++i] = c = getch()))
            ;
    s[i] = '\0';
    if (c != EOF)
        ungetch(c);
    return NUMBER;
}
```

K&R 4.3-5

```
#define BUFSIZE 100

char buf[BUFSIZE];      /* buffer for ungetch */
int bufp = 0;            /* next free position in buf */

int getch(void) /* get a (possibly pushed-back) character */
{
    return (bufp > 0) ? buf[--bufp] : getchar();
}

void ungetch(int c) /* push character back on input */
{
    if (bufp >= BUFSIZE)
        printf("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}
```

calc.h

```
#define NUMBER '0'  
void push(double);  
double pop(void);  
int getop(char []);  
int getch(void);  
void ungetch(int);
```

main.c

```
#include <stdio.h>  
#include <stdlib.h>  
#include "calc.h"  
#define MAXOP 100  
main() {  
    ...  
}
```

getop.c

```
#include <stdio.h>  
#include <ctype.h>  
#include "calc.h"  
getop() {  
    ...  
}
```

getch.c

```
#include <stdio.h>  
#define BUFSIZE 100  
char buf[BUFSIZE];  
int bufp = 0;  
int getch(void) {  
    ...  
}  
void ungetch(int) {  
    ...  
}
```

stack.c

```
#include <stdio.h>  
#include "calc.h"  
#define MAXVAL 100  
int sp = 0;  
double val[MAXVAL];  
void push(double) {  
    ...  
}  
double pop(void) {  
    ...  
}
```

K&R 4.4

```
#include <stdio.h>

/* printd:  print n in decimal */
void printd(int n)
{
    if (n < 0) {
        putchar('-');
        n = -n;
    }
    if (n / 10)
        printd(n / 10);
    putchar(n % 10 + '0');
}
```

K&R 4.5

```
/* qsort:  sort v[left]...v[right] into increasing order */
void qsort(int v[], int left, int right)
{
    int i, last;
    void swap(int v[], int i, int j);

    if (left >= right) /* do nothing if array contains */
        return;           /* fewer than two elements */
    swap(v, left, (left + right)/2); /* move partition elem */
    last = left;           /* to v[0] */
    for (i = left + 1; i <= right; i++) /* partition */
        if (v[i] < v[left])
            swap(v, ++last, i);
    swap(v, left, last);           /* restore partition elem */
    qsort(v, left, last-1);
    qsort(v, last+1, right);
}

/* swap:  interchange v[i] and v[j] */
void swap(int v[], int i, int j)
{
    int temp;

    temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}
```

K&R 5.1

```
void swap(int x, int y) /* WRONG */
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}

void swap(int *px, int *py) /* interchange *px and *py */
{
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}
```

K&R 5.2

```
#include <ctype.h>

int getch(void);
void ungetch(int);

/* getint: get next integer from input into *pn */
int getint(int *pn)
{
    int c, sign;

    while (isspace(c = getch())) /* skip white space */
        ;
    if (!isdigit(c) && c != EOF && c != '+' && c != '-')
        ungetch(c); /* it is not a number */
        return 0;
    }
    sign = (c == '-') ? -1 : 1;
    if (c == '+' || c == '-')
        c = getch();
    for (*pn = 0; isdigit(c), c = getch())
        *pn = 10 * *pn + (c - '0');
    *pn *= sign;
    if (c != EOF)
        ungetch(c);
    return c;
}
```

K&R 5.3

```
#include <ctype.h>

int getch(void);
void ungetch(int);

/* getint: get next integer from input into *pn */
int getint(int *pn)
{
    int c, sign;

    while (isspace(c = getch())) /* skip white space */
        ;
    if (!isdigit(c) && c != EOF && c != '+' && c != '-')
        ungetch(c); /* it is not a number */
        return 0;
    }
    sign = (c == '-') ? -1 : 1;
    if (c == '+' || c == '-')
        c = getch();
    for (*pn = 0; isdigit(c), c = getch())
        *pn = 10 * *pn + (c - '0');
    *pn *= sign;
    if (c != EOF)
        ungetch(c);
    return c;
}
```

K&R 5.4

```
#define ALLOCSIZE 10000 /* size of available space */

static char allocbuf[ALLOCSIZE]; /* storage for alloc */
static char *allocp = allocbuf; /* next free position */

char *alloc(int n)      /* return pointer to n characters */
{
    if (allocbuf + ALLOCSIZE - allocp >= n) { /* it fits */
        allocp += n;
        return allocp - n; /* old p */
    } else          /* not enough room */
        return 0;
}

void afree(char *p) /* free storage pointed to by p */
{
    if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
        allocp = p;
}
```

K&R 5.5

```
/* strlen:  return length of string s */
int strlen(char *s)
{
    char *p = s;

    while (*p != '\0')
        p++;
    return p - s;
}

/* strcpy:  copy t to s; array subscript version */
void strcpy(char *s, char *t)
{
    int i;

    i = 0;
    while ((s[i] = t[i]) != '\0')
        i++;
}

/* strcpy:  copy t to s; pointer version */
void strcpy(char *s, char *t)
{
    int i;

    i = 0;
    while ((*s = *t) != '\0') {
        s++;
        t++;
    }
}
```

K&R 5.6

```
/* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
    int i;

    for (i = 0; s[i] == t[i]; i++)
        if (s[i] == '\0')
            return 0;
    return s[i] - t[i];
}

/* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
    for ( ; *s == *t; s++, t++)
        if (*s == '\0')
            return 0;
    return *s - *t;
}
```

K&R 5.7-1

```
#include <stdio.h>
#include <string.h>

#define MAXLINES 5000      /* max #lines to be sorted */

char *lineptr[MAXLINES]; /* pointers to text lines */

int readlines(char *lineptr[], int nlines);
void writelines(char *lineptr[], int nlines);

void qsort(char *lineptr[], int left, int right);

/* sort input lines */
main()
{
    int nlines;      /* number of input lines read */

    if ((nlines = readlines(lineptr, MAXLINES)) >= 0) {
        qsort(lineptr, 0, nlines-1);
        writelines(lineptr, nlines);
        return 0;
    } else {
        printf("error: input too big to sort\n");
        return 1;
    }
}
```

```
#define MAXLEN 1000 /* max length of any input line */
int getline(char *, int);
char *alloc(int);

/* readlines: read input lines */
int readlines(char *lineptr[], int maxlines)
{
    int len, nlines;
    char *p, line[MAXLEN];

    nlines = 0;
    while ((len = getline(line, MAXLEN)) > 0)
        if (nlines >= maxlines || p = alloc(len) == NULL)
            return -1;
        else {
            line[len-1] = '\0'; /* delete newline */
            strcpy(p, line);
            lineptr[nlines++] = p;
        }
    return nlines;
}

/* writelines: write output lines */
void writelines(char *lineptr[], int nlines)
{
    int i;

    for (i = 0; i < nlines; i++)
        printf("%s\n", lineptr[i]);
}
```

```
/* qsort:  sort v[left]...v[right] into increasing order */
void qsort(char *v[], int left, int right)
{
    int i, last;
    void swap(char *v[], int i, int j);

    if (left >= right) /* do nothing if array contains */
        return;           /* fewer than two elements */
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
        if (strcmp(v[i], v[left]) < 0)
            swap(v, ++last, i);
    swap(v, left, last);
    qsort(v, left, last-1);
    qsort(v, last+1, right);
}
```

K&R 5.7-3

```
/* swap:  interchange v[i] and v[j] */
void swap(char *v[], int i, int j)
{
    char *temp;

    temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}
```

K&R 5.8

```
#include <stdio.h>
#include <string.h>
#define MAXLINE 1000

int getline(char *line, int max);

/* find: print lines that match pattern from 1st arg */
main(int argc, char *argv[])
{
    char line[MAXLINE];
    int found = 0;

    if (argc != 2)
        printf("Usage: find pattern\n");
    else
        while (getline(line, MAXLINE) > 0)
            if (strstr(line, argv[1]) != NULL) {
                printf("%s", line);
                found++;
            }
    return found;
}
```

K&R 5.8-1

```
#include <stdio.h>
#include <string.h>
#define MAXLINE 1000

int getline(char *line, int max);

/* find: print lines that match pattern from 1st arg */
main(int argc, char *argv[])
{
    char line[MAXLINE];
    long lineno = 0;
    int c, except = 0, number = 0, found = 0;

    while (--argc > 0 && (*++argv)[0] == '-')
        while (c = *++argv[0])
            switch (c) {
                case 'x':
                    except = 1;
                    break;
                case 'n':
                    number = 1;
                    break;
                default:
                    printf("find: illegal option %c\n", c);
                    argc = 0;
                    found = -1;
                    break;
            }
    if (found != -1)
        for (lineno = 1; ; lineno++)
            if (getline(line, MAXLINE) == 0)
                break;
            else if (number == 0 || line[0] == c)
                if (except == 0)
                    printf("%s", line);
                else if (c == 'x')
                    except = 0;
                else if (c == 'n' && line[1] == '\n')
                    except = 0;
            else
                except = 1;
}
```

K&R 5.8-2

```
if (argc != 1)
    printf("Usage: find -x -n pattern\n");
else
    while (getline(line, MAXLINE) > 0) {
        lineno++;
        if ((strstr(line, *argv) != NULL) != except) {
            if (number)
                printf("%ld:", lineno);
            printf("%s", line);
            found++;
        }
    }
return found;
}
```

K&R 5.9-1

```
#include <stdio.h>
#include <string.h>

#define MAXLINES 5000      /* max #lines to be sorted */
char *lineptr[MAXLINES]; /* pointers to text lines */

int readlines(char *lineptr[], int nlines);
void writelines(char *lineptr[], int nlines);

void qsort(void *lineptr[], int left, int right,
           int (*comp)(void *, void *));
int numcmp(char *, char *);

/* sort input lines */
main(int argc, char *argv[])
{
    int nlines;          /* number of input lines read */
    int numeric = 0;     /* 1 if numeric sort */

    if (argc > 1 && strcmp(argv[1], "-n") == 0)
        numeric = 1;
    if ((nlines = readlines(lineptr, MAXLINES)) >= 0) {
        qsort((void**) lineptr, 0, nlines-1,
               (int (*)(void*,void*))(numeric ? numcmp : strcmp));
        writelines(lineptr, nlines);
        return 0;
    } else {
        printf("input too big to sort\n");
        return 1;
    }
}
```

K&R 5.9-2

```
/* qsort:  sort v[left]...v[right] into increasing order */
void qsort(void *v[], int left, int right,
           int (*comp)(void *, void *))
{
    int i, last;

    void swap(void *v[], int, int);

    if (left >= right)      /* do nothing if array contains */
        return;              /* fewer than two elements */
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
        if ((*comp)(v[i], v[left]) < 0)
            swap(v, ++last, i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp);
}
```

K&R 5.9-3

```
#include <stdlib.h>

/* numcmp: compare s1 and s2 numerically */
int numcmp(char *s1, char *s2)
{
    double v1, v2;

    v1 = atof(s1);
    v2 = atof(s2);
    if (v1 < v2)
        return -1;
    else if (v1 > v2)
        return 1;
    else
        return 0;
}

void swap(void *v[], int i, int j;)
{
    void *temp;

    temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}
```