

SOFTWARE—Ph.D. Qualifying Exam Spring 2009

(i) (7 pts.)¹ The Sieve of Eratosthenes is a method for finding prime numbers. This method involves the following two-step process.

1. Create an array with all elements initialized to 1 (true).
2. Starting with array subscript 2, every time an array element is found whose value is 1, loop through the remainder of the array, and set to zero every element whose subscript is a multiple of the subscript for the element with value 1. For array subscript 2, all elements beyond 2 in the array that are multiples of 2 will be set to zero (subscripts 4, 6, 8, 10, etc.). For array subscript 3, all elements beyond 3 in the array that are multiples of 3 will be set to zero (subscripts 6, 9, 12, 15, etc.).

When this process is complete, the array elements that are still set to one indicate that the subscript is a prime number.

Write a valid C program that uses the Sieve of Eratosthenes method to determine and print the prime numbers between 1 and 999. Ignore element 0 of the array.

1. This problem is adapted from a problem in Deitel and Deitel, *C How to Program*, Third Edition.

Write your solution to Question i on this page. Please clearly indicate your solution and show all work on this page.

(ii) (7 pts.) Consider the following structure definitions for a linked list.

```
/* Container for one element of a linked list of strings. */
struct list_element {
    /* Pointer to the string associated with this list element.*/
    char *string;

    /* Pointer to the next element of this list. */
    struct list_element *next;
};

/* A linked list of strings */
struct list_struct {
    /* Pointer to first element of the list. If the list is empty,
       then "head" is NULL.
    */
    struct list_element *head;
};
```

Consider also the following function prototype and associated header comment.

```
/******
Insert the given string into the given list in such a way that sorted
ordering in the list is maintained. Upon entry to the function, it is
assumed that the given list is either empty or is non-empty and in
(alphabetically) sorted order. That is, the list is sorted in terms of
the strings that it contains. The function should insert the given string
into the list so that the modified list is still in sorted order after
the insertion.
*****/
void list_insert_sorted(struct list_struct *list, char *string)
```

Develop a complete C code implementation of the function `list_insert_sorted`.

Write your solution to Question ii on this page. Please clearly indicate your solution and show all work on this page.

(iii) (6 points)¹ Given a set $X = \{x_1, x_2, \dots, x_n\}$ of numbers, the geometric mean of X is given by $\sqrt[n]{x_1 \times x_2 \times \dots \times x_n}$, and the harmonic mean is given by

$$\frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \dots + \frac{1}{x_n}}$$

Write a valid C program that continues to accept numbers (use the *double* type) from standard input until the number zero (0) is entered, and then calculates and displays both the geometric and harmonic means of the entered numbers. NOTE: the terminating zero should be *excluded* from the set of numbers for which the means are calculated; the zero is used only to indicate the end of the list in standard input.

1. This problem is adapted from a problem in G. Bronson, *C for Scientists and Engineers*.

Write your solution to Question iii on this page. Please clearly indicate your solution and show all work on this page.

C Reference Card (ANSI)

Program Structure/Functions

```

type func(type1, ...);
type name;
int main(void) {
    declarations
    statements
}

type func(arg1, ... ) {
    declarations
    statements
}

/* */
int main(int argc, char *argv[])
exit(arg);

```

C Preprocessor

```

include library file
include user file
replacement text
replacement macro
Example. #define max(A, B) ((A)>(B) ? (A) : (B))

#undef name
quoted string in replace
Example. #define msg(A) printf("%s = %d", #A, (A))
concatenate args and rescan
conditional execution
is name defined, not defined?
name defined?
line continuation char

```

Data Types/Declarations

```

character (1 byte)
integer
real number (single, double precision)
short (16 bit integer)
long (32 bit integer)
double long (64 bit integer)
positive or negative
non-negative modulo 2m
pointer to int, float, ...
enumeration constant
constant (read-only) value
declare external variable
internal to source file
local persistent between calls
no value
structure
create new name for data type
size of an object (type is size_t)
size of a data type (type is size_t)

```

Initialization

```

initialize variable
initialize array
initialize char string

```

Constants

```

suffix: long, unsigned, float
exponential form
prefix: octal, hexadecimal
Example. 031 is 25, 0x31 is 49 decimal
character constant (char, octal, hex)
newline, cr, tab, backspace
special characters
string constant (ends with '\0')

```

Pointers, Arrays & Structures

```

declare pointer to type
declare function returning pointer to type type *f();
declare pointer to function returning type type (*pf)();
generic pointer type
null pointer constant
object pointed to by pointer
address of object name
array
multi-dim array
structures
struct tag {
    declarations
};
create structure
member of structure from template
member of pointed-to structure
Example. (*p).x and p->x are the same
single object, multiple possible types
bit field with b bits
unsigned member: b;

```

Operators (grouped by precedence)

```

struct member operator
struct member through pointer
increment, decrement
plus, minus, logical not, bitwise not
indirection via pointer, address of object
cast expression to type
size of an object
multiply, divide, modulus (remainder)
add, subtract
left, right shift [bit ops]
relational comparisons
equality comparisons
and [bit op]
exclusive or [bit op]
or [inclusive] [bit op]
logical and
logical or
conditional expression
assignment operators
expression evaluation separator
Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

```

Flow of Control

```

statement terminator
block delimiters
exit from switch, while, do, for
next iteration of while, do, for
goto
label
return value from function
Flow Constructions
if statement
while statement
for statement
do statement
switch statement
if (expr1) statement1
else if (expr2) statement2
else statement3
while (expr)
for (expr1; expr2; expr3)
    statement
do statement
while (expr);
switch (expr) {
    case const1: statement1 break;
    case const2: statement2 break;
    default: statement
}

```

ANSI Standard Libraries

```

<assert.h> <ctype.h> <errno.h> <float.h> <limits.h>
<local.h> <math.h> <setjmp.h> <signal.h> <stdarg.h>
<stddef.h> <stdio.h> <stdlib.h> <string.h> <time.h>

```

Character Class Tests <ctype.h>

```

alphanumeric?
alphabetic?
control character?
decimal digit?
printing character (not incl space)?
lower case letter?
printing character (incl space)?
printing char except space, letter, digit?
space, formatted, newline, cr, tab, vtab?
upper case letter?
hexadecimal digit?
convert to lower case
convert to upper case

```

String Operations <string.h>

```

s is a string; cs, ct are constant strings
length of s
copy ct to s
concatenate ct after s
compare cs to ct
only first n chars
pointer to first c in cs
pointer to last c in cs
copy n chars from ct to s
copy n chars from ct to s (may overlap)
compare n chars of cs with ct
pointer to first c in first n chars of cs
put c into first n chars of s
strlen(s)
strcpy(s,ct)
strcat(s,ct)
strcmp(cs,ct)
strncmp(cs,ct,n)
strchr(cs,c)
strrchr(cs,c)
memcpy(s,ct,n)
memmove(s,ct,n)
memcmp(cs,ct,n)
memchr(cs,c,n)
memset(s,c,n)

```

C Reference Card (ANSI)

Input/Output <stdio.h>

Standard I/O
 standard input stream
 standard output stream
 standard error stream
 end of file (type is int)
 get a character
 print a character
 print formatted data
 print to string s
 read formatted data
 read from string s
 print string s
File I/O
 declare file pointer
 pointer to named file
 modes: r (read), w (write), a (append), b (binary)
 get a character
 write a character
 write to file
 read from file
 read and store n elts to *ptr
 write n elts from *ptr to file
 close file
 non-zero if error
 non-zero if already reached EOF
 read line to string s (< max chars)
 write string s

FILE *fp;
fopen("name", "mode")
getc(fp)
putc(chr, fp)
fprintf(fp, "format", arg1, ...)
scanf(fp, "format", arg1, ...)
fread(*ptr, elsize, n, fp)
fwrite(*ptr, elsize, n, fp)
fclose(fp)
feof(fp)
fgetc(s, max, fp)
fputs(s, fp)
 + print with sign
 space print space if no sign
 0 pad with leading zeros
 w min field width
 p precision
 m conversion character:
 c short, l long, L long double
 conversion character:
 d, i integer u unsigned
 c single char s char string
 f double (printf) e, E exponential
 f float (scanf) lf double (scanf)
 o octal x, X hexadecimal
 p pointer n number of chars written
 %g, %e same as f or e, E depending on exponent

Variable Argument Lists <stdarg.h>

declaration of pointer to arguments
 initialization of argument pointer
 access next unnamed arg, update pointer
 call before exiting function

Standard Utility Functions <stdlib.h>

absolute value of int n
 quotient and remainder of ints n,d
 returns structure with div_t, quot and div_t, rem
 quotient and remainder of longs n,d
 returns structure with ldiv_t, quot and ldiv_t, rem
 pseudo-random integer [0, RAND_MAX]
 set random seed to n
 terminate program execution
 pass string s to system for execution
Conversions
 convert string s to double
 convert string s to integer
 convert string s to long
 convert prefix of s to double
 convert prefix of s (base b) to long
 same, but unsigned long
Storage Allocation
 allocate storage
 deallocate storage
Array Functions
 search array for key
 sort array ascending order

abs(n)
labs(n)
div(n,d)
ldiv(n,d)
rand()
srand(n)
exit(status)
system(s)
atof(s)
atoi(s)
atol(s)
strtod(s, &endp)
strtol(s, &endp, b)
strtoul(s, &endp, b)
malloc(size), calloc(nobj, size)
realloc(ptr, size); free(ptr);
bsearch(key, array, n, size, cmpf)
qsort(array, n, size, cmpf)
clock()
 processor time used by program
clock() / CLOCKS_PER_SEC is time in seconds
 current calendar time
 time₂ - time₁ in seconds (double)
 arithmetic types representing times
 structure type for calendar time comps
 tm_sec seconds after minute
 tm_min minutes after hour
 tm_hour hours since midnight
 tm_mday day of month
 tm_mon months since January
 tm_year years since 1900
 tm_wday days since Sunday
 tm_yday days since January 1
 tm_isdst Daylight Savings Time flag
 convert local time to calendar time
 convert time in tp to string
 convert calendar time in tp to local time
 convert calendar time to GMT
 convert calendar time to local time
 format date and time info
 tp is a pointer to a structure of type tm

Time and Date Functions <time.h>

absolute value of int n
 quotient and remainder of ints n,d
 returns structure with div_t, quot and div_t, rem
 quotient and remainder of longs n,d
 returns structure with ldiv_t, quot and ldiv_t, rem
 pseudo-random integer [0, RAND_MAX]
 set random seed to n
 terminate program execution
 pass string s to system for execution
Conversions
 convert string s to double
 convert string s to integer
 convert string s to long
 convert prefix of s to double
 convert prefix of s (base b) to long
 same, but unsigned long
Storage Allocation
 allocate storage
 deallocate storage
Array Functions
 search array for key
 sort array ascending order

Mathematical Functions <math.h>

arguments and returned values are double
 trig functions
 inverse trig functions
 arctan(y/x)
 hyperbolic trig functions
 exponentials & logs (2 power)
 division & remainder
 powers
 rounding
<limits.h>
 CHAR_MIN min value of char
 CHAR_MAX max value of char
 SCHAR_MIN min signed char
 SCHAR_MAX max signed char
 SHRT_MIN min value of short
 SHRT_MAX max value of short
 INT_MIN min value of int
 INT_MAX max value of int
 LONG_MIN min value of long
 LONG_MAX max value of long
 ULONG_MIN min value of long
 ULONG_MAX max unsigned long
 UCHAR_MIN min value of unsigned char
 UCHAR_MAX max unsigned char
 USHRT_MIN min value of unsigned short
 USHRT_MAX max unsigned short
 UINT_MIN min unsigned int
 UINT_MAX max unsigned int
 ULONG_MIN min unsigned long
 ULONG_MAX max unsigned long

sin(x), cos(x), tan(x)
asin(x), acos(x), atan(x)
atan2(y,x)
sinh(x), cosh(x), tanh(x)
exp(x), log(x), log10(x)
ldexp(x,n), frexp(x, &e)
modf(x, &ip), fmod(x,y)
pow(x,y), sqrt(x)
ceil(x), floor(x), fabs(x)
sinh(x), cosh(x), tanh(x)
exp(x), log(x), log10(x)
ldexp(x,n), frexp(x, &e)
modf(x, ip), fmod(x,y)
pow(x,y), sqrt(x)
ceil(x), floor(x), fabs(x)
(SCHAR_MAX or UCHAR_MAX)
(SCHAR_MIN or 0)
(+127)
(-128)
(+32,767)
(-32,768)
(+2,147,483,647)
(-2,147,483,648)
(+32,767)
(-32,768)
(+2,147,483,647)
(-2,147,483,648)
(255)
(65,535)
(65,535)
(4,294,967,295)
(4,294,967,295)

Float Type Limits <float.h>

The numbers given in parentheses are typical values for the constants on a 32-bit Unix system.
 FLT_RADIX radix of exponent rep
 FLT_ROUNDS floating point rounding mode
 FLT_DIG decimal digits of precision
 FLT_EPSILON smallest x so 1.0f + x ≠ 1.0f
 FLT_MANT_DIG number of digits in mantissa
 FLT_MAX maximum float number
 FLT_MIN minimum float number
 FLT_MIN_EXP minimum exponent
 FLT_MIN_EXP_10 minimum float number
 DBL_DIG decimal digits of precision
 DBL_EPSILON smallest x so 1.0 + x ≠ 1.0
 DBL_MANT_DIG number of digits in mantissa
 DBL_MAX maximum double number
 DBL_MIN minimum double number
 DBL_MIN_EXP minimum exponent

January 2007 v2.2. Copyright © 2007 Joseph H. Silverman
 Permission is granted to make and distribute copies of this card provided the copyright notice and this permission notice are preserved on all copies.
 Send comments and corrections to J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. (jhs@math.brown.edu)

char, and the return value is an int. The functions return non-zero (true) if the argument c satisfies the condition described, and zero if not.

isalnum(c)	isalpha(c) or isdigit(c) is true
isalpha(c)	isupper(c) or islower(c) is true
iscntrl(c)	control character
isdigit(c)	decimal digit
isgraph(c)	printing character except space
islower(c)	lower-case letter
isprint(c)	printing character including space
ispunct(c)	printing character except space or letter or digit
isspace(c)	space, formfeed, newline, carriage return, tab, vertical tab
isupper(c)	upper-case letter
isxdigit(c)	hexadecimal digit

In the seven-bit ASCII character set, the printing characters are 0x20 (' ') to 0x7E ('~'); the control characters are 0 (NUL) to 0x1F (US), and 0x7F (DEL).

In addition, there are two functions that convert the case of letters:

int tolower(int c)	convert c to lower case
int toupper(int c)	convert c to upper case

If c is an upper-case letter, tolower(c) returns the corresponding lower-case letter; otherwise it returns c. If c is a lower-case letter, toupper(c) returns the corresponding upper-case letter; otherwise it returns c.

B3. String Functions: <string.h>

There are two groups of string functions defined in the header <string.h>. The first have names beginning with str; the second have names beginning with mem. Except for memmove, the behavior is undefined if copying takes place between overlapping objects. Comparison functions treat arguments as unsigned char arrays.

In the following table, variables s and t are of type char *; cs and ct are of type const char *; n is of type size_t; and c is an int converted to char.

char *strcpy(s,ct)	copy string ct to string s, including '\0'; return s.
char *strncpy(s,ct,n)	copy at most n characters of string ct to s; return s. Pad with '\0's if t has fewer than n characters.
char *strcat(s,ct)	concatenate string ct to end of string s; return s.
char *strncat(s,ct,n)	concatenate at most n characters of string ct to string s, terminate s with '\0'; return s.
int strcmp(cs,ct)	compare string cs to string ct; return <0 if cs<ct, 0 if cs==ct, or >0 if cs>ct.
int strncmp(cs,ct,n)	compare at most n characters of string cs to string ct; return <0 if cs<ct, 0 if cs==ct, or >0 if cs>ct.
char *strchr(cs,c)	return pointer to first occurrence of c in cs or NULL if not present.
char *strrchr(cs,c)	return pointer to last occurrence of c in cs or NULL if not present.

<code>size_t strspn(cs,ct)</code>	return length of prefix of <i>cs</i> consisting of characters in <i>ct</i> .
<code>size_t strcspn(cs,ct)</code>	return length of prefix of <i>cs</i> consisting of characters <i>not</i> in <i>ct</i> .
<code>char *strpbrk(cs,ct)</code>	return pointer to first occurrence in string <i>cs</i> of any character of string <i>ct</i> , or NULL if none are present.
<code>char *strstr(cs,ct)</code>	return pointer to first occurrence of string <i>ct</i> in <i>cs</i> , or NULL if not present.
<code>size_t strlen(cs)</code>	return length of <i>cs</i> .
<code>char *strerror(n)</code>	return pointer to implementation-defined string corresponding to error <i>n</i> .
<code>char *strtok(s,ct)</code>	<code>strtok</code> searches <i>s</i> for tokens delimited by characters from <i>ct</i> ; see below.

A sequence of calls of `strtok(s,ct)` splits *s* into tokens, each delimited by a character from *ct*. The first call in a sequence has a non-NULL *s*. It finds the first token in *s* consisting of characters not in *ct*; it terminates that by overwriting the next character of *s* with `'\0'` and returns a pointer to the token. Each subsequent call, indicated by a NULL value of *s*, returns the next such token, searching from just past the end of the previous one. `strtok` returns NULL when no further token is found. The string *ct* may be different on each call.

The `mem..` functions are meant for manipulating objects as character arrays; the intent is an interface to efficient routines. In the following table, *s* and *t* are of type `void *`; *cs* and *ct* are of type `const void *`; *n* is of type `size_t`; and *c* is an `int` converted to an unsigned `char`.

<code>void *memcpy(s,ct,n)</code>	copy <i>n</i> characters from <i>ct</i> to <i>s</i> , and return <i>s</i> .
<code>void *memmove(s,ct,n)</code>	same as <code>memcpy</code> except that it works even if the objects overlap.
<code>int memcmp(cs,ct,n)</code>	compare the first <i>n</i> characters of <i>cs</i> with <i>ct</i> ; return as with <code>strcmp</code> .
<code>void *memchr(cs,c,n)</code>	return pointer to first occurrence of character <i>c</i> in <i>cs</i> , or NULL if not present among the first <i>n</i> characters.
<code>void *memset(s,c,n)</code>	place character <i>c</i> into first <i>n</i> characters of <i>s</i> , return <i>s</i> .

B4. Mathematical Functions: `<math.h>`

The header `<math.h>` declares mathematical functions and macros.

The macros `EDOM` and `ERANGE` (found in `<errno.h>`) are non-zero integral constants that are used to signal domain and range errors for the functions; `HUGE_VAL` is a positive `double` value. A *domain error* occurs if an argument is outside the domain over which the function is defined. On a domain error, `errno` is set to `EDOM`; the return value is implementation-dependent. A *range error* occurs if the result of the function cannot be represented as a `double`. If the result overflows, the function returns `HUGE_VAL` with the right sign, and `errno` is set to `ERANGE`. If the result underflows, the function returns zero; whether `errno` is set to `ERANGE` is implementation-defined.

In the following table, *x* and *y* are of type `double`, *n* is an `int`, and all functions return `double`. Angles for trigonometric functions are expressed in radians.

B5. U

The
tion, and
double
ato
int at
conv
long a
conv
double
str
point

<code>size_t strspn(cs,ct)</code>	return length of prefix of <i>cs</i> consisting of characters in <i>ct</i> .
<code>size_t strcspn(cs,ct)</code>	return length of prefix of <i>cs</i> consisting of characters <i>not</i> in <i>ct</i> .
<code>char *strpbrk(cs,ct)</code>	return pointer to first occurrence in string <i>cs</i> of any character of string <i>ct</i> , or NULL if none are present.
<code>char *strstr(cs,ct)</code>	return pointer to first occurrence of string <i>ct</i> in <i>cs</i> , or NULL if not present.
<code>size_t strlen(cs)</code>	return length of <i>cs</i> .
<code>char *strerror(n)</code>	return pointer to implementation-defined string corresponding to error <i>n</i> .
<code>char *strtok(s,ct)</code>	<code>strtok</code> searches <i>s</i> for tokens delimited by characters from <i>ct</i> ; see below.

A sequence of calls of `strtok(s,ct)` splits *s* into tokens, each delimited by a character from *ct*. The first call in a sequence has a non-NULL *s*. It finds the first token in *s* consisting of characters not in *ct*; it terminates that by overwriting the next character of *s* with `'\0'` and returns a pointer to the token. Each subsequent call, indicated by a NULL value of *s*, returns the next such token, searching from just past the end of the previous one. `strtok` returns NULL when no further token is found. The string *ct* may be different on each call.

The `mem..` functions are meant for manipulating objects as character arrays; the intent is an interface to efficient routines. In the following table, *s* and *t* are of type `void *`; *cs* and *ct* are of type `const void *`; *n* is of type `size_t`; and *c* is an `int` converted to an unsigned `char`.

<code>void *memcpy(s,ct,n)</code>	copy <i>n</i> characters from <i>ct</i> to <i>s</i> , and return <i>s</i> .
<code>void *memmove(s,ct,n)</code>	same as <code>memcpy</code> except that it works even if the objects overlap.
<code>int memcmp(cs,ct,n)</code>	compare the first <i>n</i> characters of <i>cs</i> with <i>ct</i> ; return as with <code>strcmp</code> .
<code>void *memchr(cs,c,n)</code>	return pointer to first occurrence of character <i>c</i> in <i>cs</i> , or NULL if not present among the first <i>n</i> characters.
<code>void *memset(s,c,n)</code>	place character <i>c</i> into first <i>n</i> characters of <i>s</i> , return <i>s</i> .

B4. Mathematical Functions: `<math.h>`

The header `<math.h>` declares mathematical functions and macros.

The macros `EDOM` and `ERANGE` (found in `<errno.h>`) are non-zero integral constants that are used to signal domain and range errors for the functions; `HUGE_VAL` is a positive `double` value. A *domain error* occurs if an argument is outside the domain over which the function is defined. On a domain error, `errno` is set to `EDOM`; the return value is implementation-dependent. A *range error* occurs if the result of the function cannot be represented as a `double`. If the result overflows, the function returns `HUGE_VAL` with the right sign, and `errno` is set to `ERANGE`. If the result underflows, the function returns zero; whether `errno` is set to `ERANGE` is implementation-defined.

In the following table, *x* and *y* are of type `double`, *n* is an `int`, and all functions return `double`. Angles for trigonometric functions are expressed in radians.

B5. U

The
tion, and
double
ato
int at
conv
long a
conv
double
str
point

characters in

characters

cs of any present.

st in cs, or

ed string

characters

imited by a
nds the first
ing the next
nt call, indi-
ust past the
found. The

arrays; the
are of type
c is an int

s.
even if the

st; return as

ter c in cs,
aracters.
s, return s.

integral con-
GE_VAL is a
the domain
M; the return
the function
tion returns
t underflows,
n-defined.
all functions

<code>sin(x)</code>	sine of x
<code>cos(x)</code>	cosine of x
<code>tan(x)</code>	tangent of x
<code>asin(x)</code>	$\sin^{-1}(x)$ in range $[-\pi/2, \pi/2]$, $x \in [-1, 1]$.
<code>acos(x)</code>	$\cos^{-1}(x)$ in range $[0, \pi]$, $x \in [-1, 1]$.
<code>atan(x)</code>	$\tan^{-1}(x)$ in range $[-\pi/2, \pi/2]$.
<code>atan2(y,x)</code>	$\tan^{-1}(y/x)$ in range $[-\pi, \pi]$.
<code>sinh(x)</code>	hyperbolic sine of x
<code>cosh(x)</code>	hyperbolic cosine of x
<code>tanh(x)</code>	hyperbolic tangent of x
<code>exp(x)</code>	exponential function e^x
<code>log(x)</code>	natural logarithm $\ln(x)$, $x > 0$.
<code>log10(x)</code>	base 10 logarithm $\log_{10}(x)$, $x > 0$.
<code>pow(x,y)</code>	x^y . A domain error occurs if $x=0$ and $y \leq 0$, or if $x < 0$ and y is not an integer.
<code>sqrt(x)</code>	\sqrt{x} , $x \geq 0$.
<code>ceil(x)</code>	smallest integer not less than x , as a double.
<code>floor(x)</code>	largest integer not greater than x , as a double.
<code>fabs(x)</code>	absolute value $ x $
<code>ldexp(x,n)</code>	$x \cdot 2^n$
<code>frexp(x, int *exp)</code>	splits x into a normalized fraction in the interval $[1/2, 1)$, which is returned, and a power of 2, which is stored in $*exp$. If x is zero, both parts of the result are zero.
<code>modf(x, double *ip)</code>	splits x into integral and fractional parts, each with the same sign as x . It stores the integral part in $*ip$, and returns the fractional part.
<code>fmod(x,y)</code>	floating-point remainder of x/y , with the same sign as x . If y is zero, the result is implementation-defined.

B5. Utility Functions: <stdlib.h>

The header <stdlib.h> declares functions for number conversion, storage allocation, and similar tasks.

```
double atof(const char *s)
    atof converts s to double; it is equivalent to strtod(s, (char**)NULL).

int atoi(const char *s)
    converts s to int; it is equivalent to (int)strtol(s, (char**)NULL, 10).

long atol(const char *s)
    converts s to long; it is equivalent to strtol(s, (char**)NULL, 10).

double strtod(const char *s, char **endp)
    strtod converts the prefix of s to double, ignoring leading white space; it stores a pointer to any unconverted suffix in *endp unless endp is NULL. If the answer
```