Name: Section: Rollno:

ESC101: Fundamentals of Computing (Mid Semester Exam)

19 September, 2014

Total Number of Pages: 20 Total Points 110

Question	Points	Score
1	4	
2	6	
3	20	
4	14	
5	20	
6	10	
7	20	
8	16	
Total:	110	

Instructions

1. Read these instructions carefully.
2. Write you name, section and roll number on all the pages of the answer book, **including the ROUGH pages**.
3. Write the answers cleanly in the space provided. Space is given for rough work in the answer book.
4. Do not exchange question books or change the seat after obtaining question paper.
5. Using pens (blue/black ink) and not pencils. Do not use red pens for answering.
6. Even if no answers are written, the answer book has to be returned back with name and roll number written.

Helpful hints

1. The questions are *not* arranged according to the increasing order of difficulty. Do a quick first round where you answer the easy ones and leave the difficult ones for the subsequent rounds.
2. All blanks may NOT carry equal marks.
3. Use the cheat sheet provided in the answer book for any doubt related to C programming (Not all topics in the cheat sheet are covered in class, Not all topics covered in the class are provided in the cheat sheet.)

Question 1. (4 points) There are 2 subparts of this question. (2 + 2)

(a) What does the following program print?

```
#include<stdio.h>
int x=1;
int main() {
    x = 3;
    {
        int x;
        {
            x = 2;
        }
        printf("%d ",x);
    }
    printf("%d\n",x);
    return 0;
}
```

(b) What does the following program print?

```
#include<stdio.h>
int x;

void f();
void g();
void h();

int main() {
    x = 2;
    f();
    g();
    return 0;
}

void f() {
    int x = 3;
    h();
}

void g() {
    x = 4;
    h();
}

void h() {
    printf("%d ",x);
}
```

Question 2. (6 points) The following program is executed with input formed by the last two digits of your roll number, (For example, if Roll No. is 14XYZ, input is YZ). What will be the output of the program?

```
#include<stdio.h>
int main(){
    int Roll_No, N, k, m, a, b, c, A[10];

    scanf("%d", &Roll_No); /* Last 2 digits of YOUR ROLL NO.*/
    printf("Last two digits of Roll No. : %d\n", Roll_No);

    N = (Roll_No/10 + Roll_No%10) % 10;
    if (N < 5) {
        N = 10;
    }
    printf("Value of N: %d\n", N);

    a = 0;
    for(k = 1; k <= N; k++) { a = a + k; }
    printf("Value of a: %d\n", a);

    b = 1; m = 1;
    while(m < N/2) {
        b = b * m;
        m++;
    }
    printf("Value of b: %d\n", b);

    m = 1; k = 0;
    do {
        if( m%N == 0) {
            A[k] = m;
            k++;
        }
        m++;
    } while(k < 10);

    c = A[2];
    printf("Value of c: %d\n", c);

    return 0;
}
```

Output of the above program for **last 2 digits of your roll number** is:

Last two digits of Roll No. : _____

Value of N: _____

Value of a: _____

Value of b: _____

Value of c: _____

Question 3. (20 points) What does the following program print?

```
#include <stdio.h>
int foo(int);
int bar(int);

int main(){
    printf("%d\n", foo( bar( foo(2)*foo(4) ) ) );
    return 0;
}

int is_prime(int n)
{ /* checks if n is a prime >= 3 */
    int j;
    for (j=3; j*j <= n; j = j+2){
        if (n%j == 0) break;
    }
    return (j*j > n);
}

int foo(int n){
    int i = 3, a = 1;
    if (n==1)
        return 2;

    do {
        a = a + is_prime(i);
        if (a == n) break;
        i = i + 2;
    } while (1);

    printf("%d\n", i);
    return i;
}

int bar(int n){
    int s=0;
    while(n){
        s = s*10 + n%10;
        n = n/10;
    }

    printf("%d\n", s);
    return s;
}
```

Question 4. (14 points) You have two arrays containing numbers in sorted (non-decreasing) order. You want to combine the two separate arrays into one sorted array. For example: if array A={1,4,6,9} and array B={2,3,7}, the combined array C={1,2,3,4,6,7,9}

We have given the partial implementation of the combine function that takes two sorted arrays A (of length n) and B (of length m), and combine these into array C. Fill in the blanks to complete the function.

(Partial Implementation is on the next page.)

```
/* n is the size of array A, m is the size of array B */
void combine(int A[], int n, int B[], int m, int C[])
{
    /* a_index is the index to traverse in array A, b_index
     * for B.  c_index is for C to fill the array. Start
     * with the first element in each of A, B and C. */
    int a_index=0, b_index=0, c_index=0;

    /* Make sure, we do not overflow */
    while((_____ < n) && (_____ < m)) {
        /* Compare the elements at the current poition.
         * Pick the smaller one, and put into C. Then move to the
         * next element in the smaller element's array */
        if(A[a_index]<=B[b_index])
        {
            C[c_index] = _____;
            c_index++;
            _____;
        }
        else
        {
            C[c_index] = _____;
            _____;
            b_index++;
        }
    }

    /* At this point, we must have traversed one array
     * completely.  Traverse the other array completely and
     * copy the elements to C */
    if(b_index < m) {
        while(_____) {
            _____ = B[b_index];
            _____;
            _____;
        }
    }
    else if(a_index < n) {
        while(_____) {
            _____ = A[a_index];
            _____;
            _____;
        }
    }
    return;
}
```

Question 5. (20 points) What does the following program print?

```
#include <stdio.h>

long factorial(int);
void foo(int[], int);
float bar(int[], int);

int main()
{
    int n = 5;
    int a[6];
    foo(a,n);
    printf("%.2f\n", bar(a,n));
    return 0;
}

void foo(int a[], int n){
    int i,j;
    int b[6];
    for ( i = 0 ; i < n ; i++ ){
        for ( j = 0 ; j <= ( n - i - 2 ) ; j++ )
            printf(" "); /*print a SINGLE SPACE*/

        for( j = 0 ; j <= i ; j++ ){
            b[j]=factorial(i)/(factorial(j)*factorial(i-j));
            printf("%d ",b[j]); /*print INT followed by a SPACE*/
        }
        printf("\n");

        a[i]=(int)bar(b,i+1);
    }
}

float bar(int a[], int n){
    if (n%2 == 0)
        return (a[n/2] + a[n/2-1])/2.0;

    return (float)a[n/2] ;
}

/* compute factorial of n >= 0 */
long factorial(int n)
{
    int c;
    long result = 1;

    for( c = 1 ; c <= n ; c++ )
        result = result*c;

    return result;
}
```

Question 6. (10 points) What is the output of following program for the given inputs:

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

int main(){
    char base[] = "A Quick Brown Fox Jumps Over The Lazy Dog";
    int arr[26];

    int i, k, j=0;
    for(i = 0; i < 26; i++) {
        arr[i] = 0; /* Initialize array */
    }

    while(base[j] != '\0') { /* till end of base string */
        if((base[j] >= 'a' && base[j] <= 'z')
           || (base[j] >= 'A' && base[j] <= 'Z')) {
            base[j] = tolower(base[j]); /* convert to lower case */
            int tmp = arr[base[j] - 'a'];
            arr[base[j] - 'a'] = tmp + 1;
        }
        j++;
    }

    int num;
    scanf("%d",&num); /* the number of characters to read */
    for(k = 0;k<num;k++){
        char ch;
        scanf("%c",&ch);  /* input the character */
        ch = tolower(ch); /* convert to lower case */
        printf("%d", arr[ch-'a']); /* print something... */
    }

    return 0;
}
```

INPUT
4abcd

OUTPUT

INPUT
5uoiea

OUTPUT

Note: "ctype.h" contains declaration of function `tolower` that converts given uppercase letter to lowercase.

Question 7. (20 points) What is the output of following program for the given inputs:

```
#include <stdio.h>

const int true=1;
const int false=0;

int main() {
    int a[11];
    int b[11];
    int n;

    int i=0;
    int inc=true;
    int dec=false;
    int tog=0;

    a[0] = 0;
    b[0] = inc;

    scanf("%d", &n); /* Assume n <= 10 */
    for (i=1; i<=n; i++){
        scanf("%d",&a[i]);
    }

    for (i=1; i<=n; i++){
        if( inc && a[i]<a[i-1] ){
            tog++;
            dec=inc;
            inc=false;
        }
        else if (dec && a[i]>a[i-1]){
            tog++;
            inc=dec;
            dec=false;
        }
        b[i]=inc;
        printf("%d %d\n", inc, dec);
    }

    printf("%d\n", tog);
    for( i=0; i<n ; i++){
        printf("%d ",b[i] );
    }

    return 0;
}
```

INPUT

6

1 3 5 1 10 2

OUTPUT

INPUT

4

-1 1 -1 1

OUTPUT

Question 8. (16 points) (**NOTE:** This question has a very lengthy description and cryptic program. My advice is to attempt it after you have looked at all other questions.)

A wrap around array or cyclic array of size S is one in which indices beyond the range [0,...,S-1] are also acceptable. Indices S, S+1, S+2, ... point to the same elements as 0, 1, 2, ... and similarly negative indices are also valid, where indices -1, -2, -3 .... point to S-1, S-2, S-3,... and so on.

Consider a cyclic array of characters of size S = 8. Associated with this array are two special numbers, called as leap forward number (denoted as F) and leap backward number (denotes as B). Whenever leap_forward() function is called, all elements in the array leap forward by a step of size F, i.e., for all i, element at index i goes to its new index i+F. Similarly, whenever leap_backward() function is called, all elements in the array leap backward by step of size B (element at index i goes to index i-B).

The program given below first accepts two positive integers F and B from the user. Then program accepts a sequence of S characters from user. The string of characters must be stored in the same order. Whenever user enters a vowel, leap_forward() function is called. Similarly, whenever the user enters a consonant, leap_backward() function is called. After the user has entered all the characters, the starting point of the array and the array itself is printed.

Fill in the blanks in the program given below. Read the comments carefully as they may contain helpful instructions.

```
#include<stdio.h>

const int S=8; /* max elements */
int start = 0; /* store the position of first character in the array.*/

void leap_forward(char Arr[], int size, int step) {
    int i, j; char tmp;
    for(i=0; i<step; i++) { /* leap one step at a time */
        tmp = Arr[(start+size)%S];
        for(j=start+size-1; j>=start; j--)
            Arr[(j+1)%S] = Arr[j%S];
        Arr[________________] = tmp;
        start = ________________;
    }
}

void leap_backward(char Arr[], int size, int step) {
    int i, j; char tmp;
    for(i=0; i<step; i++) { /* leap one step at a time */
        /* we make sure we use MOD (%) operator with
         * *positive arguments only* to avoid any surprises
         * due to different compilers. */
        tmp = Arr[(start+S-1)%S];
        for(j=start; j<________________; j++)
            Arr[(j+S-1)%S] = Arr[j%S];
        Arr[________________] = tmp;
        start = (________________)%S;
    }
}

/* Continued on next page ... */

```

```
33
34 _____ isVowel(_____) {
35     return ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u';
36 }
37
38 int main(){
39     char A[S], ch; /* S characters to be stored in array */
40     int F, B, i;
41     int count=0; /* count of chracters read so far */
42     /* Values of F and B from user.
43      * Assume:  (i) F and B are positive.
44      *          (ii) 0 ≤ F ≤ S, 0 ≤ B ≤ S
45      * Make sure we *EAT AWAY* the newline */
46     scanf("%d %d\n", &F, &B);
47     while(count < S) { /* Read characters. */
48         scanf("%c", &ch); /*Assume no whitespace between characters*/
49         A[(start+count)%S] = ch; /*Store the character in the array*/
50         count++; /*one more character read.*/
51
52         /* Leap forward for vowel, backward for consonant */
53         if(_____)   { leap_forward (A, count, F); }
54         else        { leap_backward(A, count, B); }
55     }
56
57     /* Now print the start, and the string from start */
58     printf("start = %d\n", start);
59     for(i = start; i < _____; i++) {
60         putchar(A[_____]);
61     }
62     return 0;
63 }
```

C Reference Card (ANSI)

Program Structure/Functions

type fnc(*type*$_1$, ...);	function prototype
type name;	variable declaration
`int main(void) {`	main routine
declarations	local variable declarations
statements	
`}`	
type fnc(*arg*$_1$, ...) {	function definition
declarations	local variable declarations
statements	
`return` *value*;	
`}`	
`/* */`	comments
`int main(int argc, char *argv[])`	main with args
`exit(`*arg*`);`	terminate execution

C Preprocessor

include library file	`#include <`*filename*`>`
include user file	`#include "`*filename*`"`
replacement text	`#define` *name text*
replacement macro	`#define` *name*`(`*var*`)` *text*
Example. `#define max(A,B) ((A)>(B) ? (A) : (B))`	
undefine	`#undef` *name*
quoted string in replace	`#`
Example. `#define msg(A) printf("%s = %d", #A, (A))`	
concatenate args and rescan	`##`
conditional execution	`#if, #else, #elif, #endif`
is *name* defined, not defined?	`#ifdef, #ifndef`
name defined?	`defined(`*name*`)`
line continuation char	`\`

Data Types/Declarations

character (1 byte)	`char`
integer	`int`
real number (single, double precision)	`float, double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
double long (64 bit integer)	`long long`
positive or negative	`signed`
non-negative modulo 2^m	`unsigned`
pointer to `int`, `float`,...	`int*, float*,...`
enumeration constant	`enum` *tag* {*name*$_1$`=`*value*$_1$`,...};`
constant (read-only) value	*type* `const` *name*`;`
declare external variable	`extern`
internal to source file	`static`
local persistent between calls	`static`
no value	`void`
structure	`struct` *tag* {...}`;`
create new name for data type	`typedef` *type name*`;`
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof(`*type*`)`

Initialization

initialize variable	*type name*`=`*value*`;`
initialize array	*type name*`[]={`*value*$_1$`,...};`
initialize char string	`char` *name*`[]="`*string*`";`

Constants

suffix: long, unsigned, float	`65536L, -1U, 3.0F`
exponential form	`4.2e1`
prefix: octal, hexadecimal	`0, 0x` or `0X`
Example. `031` is 25, `0x31` is 49 decimal	
character constant (char, octal, hex)	`'a', '\`*ooo*`', '\x`*hh*`'`
newline, cr, tab, backspace	`\n, \r, \t, \b`
special characters	`\\, \?, \', \"`
string constant (ends with `'\0'`)	`"abc...de"`

Pointers, Arrays & Structures

declare pointer to *type*	*type* `*`*name*`;`
declare function returning pointer to *type*	*type* `*f();`
declare pointer to function returning *type*	*type* `(*pf)();`
generic pointer type	`void *`
null pointer constant	`NULL`
object pointed to by *pointer*	`*`*pointer*
address of object *name*	`&`*name*
array	*name*`[`*dim*`]`
multi-dim array	*name*`[`*dim*$_1$`][`*dim*$_2$`]...`

Structures

`struct` *tag* `{`	structure template
declarations	declaration of members
`};`	

create structure	`struct` *tag name*
member of structure from template	*name*`.`*member*
member of pointed-to structure	*pointer* `->` *member*
Example. `(*p).x` and `p->x` are the same	
single object, multiple possible types	`union`
bit field with *b* bits	`unsigned` *member*`:` *b*`;`

Operators (grouped by precedence)

struct member operator	*name*`.`*member*
struct member through pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
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	*expr*$_1$ `?` *expr*$_2$ `:` *expr*$_3$
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`	`break;`
next iteration of `while`, `do`, `for`	`continue;`
go to	`goto` *label*`;`
label	*label*`:` `statement`
return value from function	`return` *expr*

Flow Constructions

`if` statement	`if (`*expr*$_1$`)` *statement*$_1$ `else if (`*expr*$_2$`)` *statement*$_2$ `else` *statement*$_3$
`while` statement	`while (`*expr*`)` *statement*
`for` statement	`for (`*expr*$_1$`;` *expr*$_2$`;` *expr*$_3$`)` *statement*
`do` statement	`do` *statement* `while(`*expr*`);`

`switch` statement	`switch (`*expr*`) {`
	`case` *const*$_1$`:` *statement*$_1$ `break;`
	`case` *const*$_2$`:` *statement*$_2$ `break;`
	`default:` *statement*
	`}`

ANSI Standard Libraries

`<assert.h>`	`<ctype.h>`	`<errno.h>`	`<float.h>`	`<limits.h>`
`<locale.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?	`isalnum(c)`
alphabetic?	`isalpha(c)`
control character?	`iscntrl(c)`
decimal digit?	`isdigit(c)`
printing character (not incl space)?	`isgraph(c)`
lower case letter?	`islower(c)`
printing character (incl space)?	`isprint(c)`
printing char except space, letter, digit?	`ispunct(c)`
space, formfeed, newline, cr, tab, vtab?	`isspace(c)`
upper case letter?	`isupper(c)`
hexadecimal digit?	`isxdigit(c)`
convert to lower case	`tolower(c)`
convert to upper case	`toupper(c)`

String Operations `<string.h>`

`s` is a string; `cs`, `ct` are constant strings

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

C Reference Card (ANSI)

Input/Output <stdio.h>

Standard I/O

standard input stream	stdin
standard output stream	stdout
standard error stream	stderr
end of file (type is int)	EOF
get a character	getchar()
print a character	putchar(*chr*)
print formatted data	printf("*format*",arg_1,...)
print to string s	sprintf(s,"*format*",arg_1,...)
read formatted data	scanf("*format*",&$name_1$,...)
read from string s	sscanf(s,"*format*",&$name_1$,...)
print string s	puts(s)

File I/O

declare file pointer	FILE *fp;
pointer to named file	fopen("*name*","*mode*")
modes: r (read), w (write), a (append), b (binary)	
get a character	getc(*fp*)
write a character	putc(*chr*,*fp*)
write to file	fprintf(*fp*,"*format*",arg_1,...)
read from file	fscanf(*fp*,"*format*",arg_1,...)
read and store n elts to *ptr	fread(*ptr,eltsize,n,*fp*)
write n elts from *ptr to file	fwrite(*ptr,eltsize,n,*fp*)
close file	fclose(*fp*)
non-zero if error	ferror(*fp*)
non-zero if already reached EOF	feof(*fp*)
read line to string s (< max chars)	fgets(s,max,*fp*)
write string s	fputs(s,*fp*)

Codes for Formatted I/O: "%-+ 0*w.pmc*"

-	left justify
+	print with sign
space	print space if no sign
0	pad with leading zeros
w	min field width
p	precision
m	conversion character: h short, l long, L long double
c	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,G	same as f or e,E depending on exponent		

Variable Argument Lists <stdarg.h>

declaration of pointer to arguments	va_list *ap*;
initialization of argument pointer	va_start(*ap*,*lastarg*);
lastarg is last named parameter of the function	
access next unnamed arg, update pointer	va_arg(*ap*,*type*)
call before exiting function	va_end(*ap*);

Standard Utility Functions <stdlib.h>

absolute value of int n	abs(n)
absolute value of long n	labs(n)
quotient and remainder of ints n,d	div(n,d)
returns structure with div_t.quot and div_t.rem	
quotient and remainder of longs n,d	ldiv(n,d)
returns structure with ldiv_t.quot and ldiv_t.rem	
pseudo-random integer [0,RAND_MAX]	rand()
set random seed to n	srand(n)
terminate program execution	exit(status)
pass string s to system for execution	system(s)

Conversions

convert string s to double	atof(s)
convert string s to integer	atoi(s)
convert string s to long	atol(s)
convert prefix of s to double	strtod(s,&endp)
convert prefix of s (base b) to long	strtol(s,&endp,b)
same, but unsigned long	strtoul(s,&endp,b)

Storage Allocation

allocate storage	malloc(size), calloc(nobj,size)
change size of storage	newptr = realloc(ptr,size);
deallocate storage	free(ptr);

Array Functions

search array for key	bsearch(key,array,n,size,cmpf)
sort array ascending order	qsort(array,n,size,cmpf)

Time and Date Functions <time.h>

processor time used by program	clock()
Example. clock()/CLOCKS_PER_SEC is time in seconds	
current calendar time	time()
$time_2$-$time_1$ in seconds (double)	difftime($time_2$,$time_1$)
arithmetic types representing times	clock_t,time_t
structure type for calendar time comps	struct tm

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	mktime(tp)
convert time in tp to string	asctime(tp)
convert calendar time in tp to local time	ctime(tp)
convert calendar time to GMT	gmtime(tp)
convert calendar time to local time	localtime(tp)
format date and time info	strftime(s,smax,"*format*",tp)
tp is a pointer to a structure of type tm	

Mathematical Functions <math.h>

Arguments and returned values are double

trig functions	sin(x), cos(x), tan(x)
inverse trig functions	asin(x), acos(x), atan(x)
arctan(y/x)	atan2(y,x)
hyperbolic trig functions	sinh(x), cosh(x), tanh(x)
exponentials & logs	exp(x), log(x), log10(x)
exponentials & logs (2 power)	ldexp(x,n), frexp(x,&e)
division & remainder	modf(x,ip), fmod(x,y)
powers	pow(x,y), sqrt(x)
rounding	ceil(x), floor(x), fabs(x)

Integer Type Limits <limits.h>

The numbers given in parentheses are typical values for the constants on a 32-bit Unix system, followed by minimum required values (if significantly different).

CHAR_BIT	bits in char	(8)
CHAR_MAX	max value of char	(SCHAR_MAX or UCHAR_MAX)
CHAR_MIN	min value of char	(SCHAR_MIN or 0)
SCHAR_MAX	max signed char	(+127)
SCHAR_MIN	min signed char	(−128)
SHRT_MAX	max value of short	(+32,767)
SHRT_MIN	min value of short	(−32,768)
INT_MAX	max value of int	(+2,147,483,647) (+32,767)
INT_MIN	min value of int	(−2,147,483,648) (−32,767)
LONG_MAX	max value of long	(+2,147,483,647)
LONG_MIN	min value of long	(−2,147,483,648)
UCHAR_MAX	max unsigned char	(255)
USHRT_MAX	max unsigned short	(65,535)
UINT_MAX	max unsigned int	(4,294,967,295) (65,535)
ULONG_MAX	max unsigned long	(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	(2)
FLT_ROUNDS	floating point rounding mode	
FLT_DIG	decimal digits of precision	(6)
FLT_EPSILON	smallest x so $1.0\text{f} + x \neq 1.0\text{f}$	($1.1\text{E}-7$)
FLT_MANT_DIG	number of digits in mantissa	
FLT_MAX	maximum float number	($3.4\text{E}38$)
FLT_MAX_EXP	maximum exponent	
FLT_MIN	minimum float number	($1.2\text{E}-38$)
FLT_MIN_EXP	minimum exponent	
DBL_DIG	decimal digits of precision	(15)
DBL_EPSILON	smallest x so $1.0 + x \neq 1.0$	($2.2\text{E}-16$)
DBL_MANT_DIG	number of digits in mantissa	
DBL_MAX	max double number	($1.8\text{E}308$)
DBL_MAX_EXP	maximum exponent	
DBL_MIN	min double number	($2.2\text{E}-308$)
DBL_MIN_EXP	minimum exponent	

Send comments and corrections to J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. ⟨jhs@math.brown.edu⟩