Table of Contents

[// test.c 2](#_Toc182347205)

[*//* types.c 3](#_Toc182347206)

[*//* floating\_values.c 4](#_Toc182347207)

[// *constants.c* 5](#_Toc182347208)

[*//* io\_formats.c 7](#_Toc182347209)

[*//* functions.c 9](#_Toc182347210)

[*//* functions\_io.c 12](#_Toc182347211)

[// printf\_scanf.c 14](#_Toc182347212)

[*//* printf\_scanf2.c 17](#_Toc182347213)

[*//* printf\_scanf3.c 18](#_Toc182347214)

[*//* if\_statements.c 19](#_Toc182347215)

[// type\_conversions.c 22](#_Toc182347216)

[// switch\_case.c 24](#_Toc182347217)

# // test.c

// this file has the only main function and it calls a number of functions. Execution starts from here

// the other functions are listed in separate files. And attached to this document in the given order.

// these are the files you can add to your project when using IDE.

// if you are using mingw compile and link them together

// gcc –o test test.c floating\_values.c ...

#include <stdio.h>  
  
void types();  
void floating\_values();  
void constants();  
void io\_formats();  
void functions();  
void functions\_io();  
void printf\_scanf();  
void printf\_scanf2();  
void printf\_scanf3();  
void if\_statements();  
void type\_conversions();

void switch\_case();  
  
  
int main()  
{  
 types();  
 floating\_values();  
 constants();  
 io\_formats();  
 functions();  
 functions\_io();  
 printf\_scanf();  
 *//// printf\_scanf2(); use another compiler for the code of this function  
 //// i.e: https://www.tutorialspoint.com/compile\_c\_online.php for this function* printf\_scanf3();  
 if\_statements();  
 type\_conversions();  
 switch\_case();  
  
 return 0;  
}

# *//* types.c

*// this file is the value ranges of the types in C*

#include <stdio.h>  
void types()  
{  
 printf("\n-----types\n");  
 double a = 123123123.00;  
 double b = 12.3456789;  
 double c = 2312312312.123123;  
  
 printf("%lf\n", a);  
 printf("%lf\n", b);  
 printf("%lf\n", c);  
  
 *// 123123123.000000  
 // 12.345679  
 // 2312312312.123123*}  
  
*//NOTES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*/\*  
  
 Length(byte)  
type (DOS / UNIX) Range Format Spec  
  
signed char 1 -128 127 %c  
unsigned char 1 0 255 %c  
signed short int 2 -32.768 32.767 %hd  
unsigned short int 2 0 65.535 %hu  
  
signed int 2 -32.768 32.767 %d or %i  
 4 -2.147.483.648 2.147.483.647  
  
unsigned int 2 0 65.535 %u  
 4 0 4.294.967.295  
  
long int 4 -2.147.483.648 2.147.483.647 %ld or %li  
unsigned long int 4 0 4.294.967.295 %lu  
  
long long int 8 -(2^63) (2^63)-1 %lld  
unsigned long long int 8 0 (2^64)-1 %llu  
  
  
FLOATING TYPES  
 Length(byte)  
type (DOS / UNIX) min/max  
  
float 4 1.2 x 10^-38 3.4 x 10^38 %f  
double 8 1.7 x 10^-308 1.7 x 10^308 %lf  
long double 16 3.4 x 10^-4932 1.1 x 10^4932 %Lf  
  
\*/  
//https://www.geeksforgeeks.org/data-types-in-c/?ref=shm

# *//* floating\_values.c

*// this file is about how to represent floating values in computers*

#include <stdio.h>  
void floating\_values()  
{  
 printf("\n-----floating\_values\n");  
 float number = -1.3501e-015;  
  
 printf("%25.20f\n",-1.3501e-015);  
 printf("%25.20f\n",3.501e-016);  
 printf("%25.20f\n",1e-015);  
  
 if (number + (3.501e-016 +1e-015) == 0)  
 printf("floating point representations are precise");  
 else  
 printf("floating point representations are not precise");  
}  
  
*//NOTES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
  
/\*  
i.e 4 byte representation:  
32 bits will be separated to three parts  
  
1.sign bit (1 bit) S  
 1 for negative numbers, 0 for positive numbers  
2. exponent) (8 bit) EEEEEEEE  
3. fraction) (23 bit) FFFFFFFFFFFFFFFFFFFFFFF  
  
S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF  
31 30-----------23 22-------------------------------------0  
  
If E = 255 and F is a non-zero value, then V is NaN (Not a Number) and is not considered a real number. Example:  
  
0 11111111 00001000000100000000000 = Not a number  
1 11111111 00010101010001001010101 = Not a number  
  
If E = 255, F = 0, and S = 1, then V = -infinity  
If E = 255, F = 0, and S = 0, then V = +infinity  
  
If 0 < E < 255, then...  
V = (-1)^S \* 2^(E - 127) \* (1.F)  
  
First, add "1." to the beginning of the fraction part of the number. Then, multiply this number by 2^(E - 127) to adjust the decimal point's position. The portion after the decimal point is obtained by multiplying by the increasing negative powers of 2. Examples:  
  
0 10000000 00000000000000000000000 = +1 \* 2^(128 - 127) \* 1.0  
        = 2 \* (1.0)\_2  
        = 2  
  
1 10000001 10100000000000000000000 = -1 \* 2 (129 - 127) \* 1.101  
 = -2^2 \* (1.101)\_2  
 = -4(1+1/2+1/8)  
 = -6.5  
\*/*

# // *constants.c*

*// this file is about the use of different types of constants,*

*// and how to display values with scientific notation (i.e: 1.7 x 10^-2)*

#include <stdio.h>  
void constants()  
{  
  
 printf("\n%-50s\n","-----constants\n");  
 char number1 = 100; *// 0x64, 0b01100100* char number2 = -100; *//0x9c, 0b10011100  
 // regarding 16,32 bit registers  
 // 0x9c can be 0xff9c 0xffffff9c* printf("%d\n", number1); *// 100* printf("%o\n", number1); *// 144* printf("%x\n", number1); *// 64* printf("%d\n", number2); *// -100* printf("%x\n", number2); *// ffffff9c* printf("%c\n",48); *//0 ASCII characters* printf("%c\n",65); *//A* printf("%c\n",97); *//a* printf("%c\n",179); *//│* printf("%c\n",225); *//ß  
  
 // -15000 (signed) int constant  
 // 15000U (unsigned) int constant.  
 // 1200L (signed) long constant  
 // 1200Lu (unsigned) long constant.  
  
 // 123ul, 123UL, 123Lu, 123lu all the same  
  
 // 0x12 hex rep. (int) constant.  
 // 0X12L hex rep. long int constant.  
 // 0x1934ul hex rep. unsigned long int constant.  
 // 01234 octal rep. int constant  
 // 0567L octal rep. long int constant  
 // 0777U octal rep. unsigned int constant  
 // 0452Lu octal rep. unsigned long int constant.  
  
 // 1.31F  
 // 10.F  
 // -2.456f  
 // float constants  
  
 // -24.5 double constant  
 // 1.34L long double constant  
 // 10.2L long double constant  
  
 // 2.3e+04f scientific format for floating point constant  
 // 1.74e-6F scientific format for floating point constant  
 // 8.e+9f scientific format for floating point constant  
  
 // 1.34E-2f == 0.0134  
 // -1.2E+2F == 120.f* printf("%u\n" ,1234u); *// unsigned integer* printf("%d\n" ,1234); *// signed integer* printf("%i\n" ,1234); *// signed integer* printf("%ld\n", 1234L); *// signed long integer* printf("%li\n", 1234L); *// signed long integer* printf("%lu\n", 1234L); *// unsigned long integer  
 // 1234  
 // 1234  
 // 1234  
 // 1234  
 // 1234  
 // 1234* printf("%Lf\n",1.2345678L); *// long double* printf("%lf\n",1.2345678); *// double* printf("%f\n" ,1.2345678f); *// float  
 // 1.234568  
 // 1.234568  
 // 1.234568* printf("%e\n", 1.2345678e-2f); *// scientific notation* printf("%E\n", 1.2345678e-2f);  
 printf("%g\n", 1.2345678e-2f);  
 printf("%G\n", 1.2345678e-2f);  
 *// 1.234568e-02  
 // 1.234568E-02  
 // 0.0123457  
 // 0.0123457  
 // with G or g format specifier compiler  
 // decides whether to use fixed or scientific notation  
 // %g will take a number that could be represented as %f  
 // (a simple float or double) or %e (scientific notation)  
 // and return it as the shorter of the two.  
  
  
 // very small numbers* printf("%e\n", 1.2345678e-9f);  
 printf("%E\n", 1.2345678e-9f);  
 printf("%g\n", 1.2345678e-9f);  
 printf("%G\n", 1.2345678e-9f);  
 *// 1.234568e-09  
 // 1.234568E-09  
 // 1.23457e-09  
 // 1.23457E-09  
  
 // very large numbers* printf("%e\n", 1.2345678e+9);  
 printf("%E\n", 1.2345678e+9f);  
 printf("%g\n", 1.2345678e+9f);  
 printf("%G\n", 1.2345678e+9f);  
 *// 1.234568e+09  
 // 1.234568E+09  
 // 1.23457e+09  
 // 1.23457E+09* printf("%s\n", "this is a C course");  
}

# *//* io\_formats.c

*// this file includes the max-min values of the ranges for different data types also printf() functions' format specifier list  
// check also types.c, printf\_scanf.c, constants.c*

#include <stdio.h>  
#include <limits.h> *//this header file has limit information of built in types*void io\_formats()  
{  
  
 printf("\n-----io\_formats\n");  
 printf("%x\n%d\n",0x1C205470, 0x1C205470 );  
 printf("%ll\n",sizeof(0x1C205470));  
  
 printf("The value of CHAR\_BIT: %d\n", CHAR\_BIT);  
 printf("The value of SCHAR\_MIN: %d\n", SCHAR\_MIN);  
 printf("The value of SCHAR\_MAX: %d\n", SCHAR\_MAX);  
 printf("The value of UCHAR\_MAX: %u\n", UCHAR\_MAX);  
 printf("The value of CHAR\_MIN: %d\n", CHAR\_MIN);  
 printf("The value of CHAR\_MAX: %d\n", CHAR\_MAX);  
 printf("The value of MB\_LEN\_MAX: %d\n", MB\_LEN\_MAX);  
 printf("The value of SHRT\_MIN: %d\n", SHRT\_MIN);  
 printf("The value of SHRT\_MAX: %d\n", SHRT\_MAX);  
 printf("The value of USHRT\_MAX: %u\n", USHRT\_MAX);  
 printf("The value of INT\_MIN: %d\n", INT\_MIN);  
 printf("The value of INT\_MAX: %d\n", INT\_MAX);  
 printf("The value of UINT\_MAX: %u\n", UINT\_MAX);  
 printf("The value of LONG\_MIN: %ld\n", LONG\_MIN);  
 printf("The value of LONG\_MAX: %ld\n", LONG\_MAX);  
 printf("The value of ULONG\_MAX: %lu\n", ULONG\_MAX);  
  
 *// The value of CHAR\_BIT: 8  
 // The value of SCHAR\_MIN: -128  
 // The value of SCHAR\_MAX: 127  
 // The value of UCHAR\_MAX: 255  
 // The value of CHAR\_MIN: -128  
 // The value of CHAR\_MAX: 127  
 // The value of MB\_LEN\_MAX: 5  
 // The value of SHRT\_MIN: -32768  
 // The value of SHRT\_MAX: 32767  
 // The value of USHRT\_MAX: 65535  
 // The value of INT\_MIN: -2147483648  
 // The value of INT\_MAX: 2147483647  
 // The value of UINT\_MAX: 4294967295  
 // The value of LONG\_MIN: -2147483648  
 // The value of LONG\_MAX: 2147483647  
 // The value of ULONG\_MAX: 4294967295  
  
 // or is long int is represented with 8 bits  
  
 // The value of LONG\_MIN: -9223372036854775808  
 // The value of LONG\_MAX: 9223372036854775807  
 // The value of ULONG\_MAX: 18446744073709551615* printf("%d\n", sizeof(char)); *//1* printf("%d\n", sizeof(short)); *//2* printf("%d\n", sizeof(int)); *//2* printf("%d\n", sizeof(long)); *//4 (for my computer)* printf("%d\n", sizeof(float)); *//4* printf("%d\n", sizeof(double)); *//8* printf("%d\n", sizeof(long double)); *//16*}  
  
*//NOTES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*  
  
*// %c For character type.  
// %d For signed integer type.  
// %e or %E For scientific notation of floats.  
// %f For float type.  
// %g or %G For float type with the current precision.  
// %i signed integer  
// %ld or %li Long integer  
// %lf Double  
// %Lf Long double  
// %lu Unsigned int or unsigned long  
// %lli or %lld Long long  
// %llu Unsigned long long  
// %o Octal representation  
// %p Pointer  
// %s String  
// %u Unsigned int  
// %x or %X Hexadecimal representation  
// %n Prints nothing  
// %% Prints % character*

# *//* functions.c

*// this file is about the use of functions in C*

#include <stdio.h> */\*  
"For the functions we write ourselves, we use the term 'definition.'  
 The general form of a function definition in C is as follows:  
  
[Return type] <function name> ([parameters]) { ... ... }"  
\*/  
  
// ==========prototypes  
//power(int number, int pwr);*int power(int number, int pwr);  
void message();  
void message2(void);  
  
void functions()  
{  
 int n = 10;  
 int pw = 5;  
 printf("\n-----functions\n");  
 *// "Functions can only be called from within defined functions.  
 // Functions cannot be called from outside blocks."* int result = power(n,pw);  
 printf("%d. power of %d is %d\n", pw, n, result);  
 message(1.0);  
 message2();  
}  
  
*// message(1.0); X Functions cannot be called from outside blocks."  
  
// ==========function implementations  
  
//power(int number, int pwr)  
// if there is no return type indicated by default it is int  
// as if it is-> int power(int number, int pwr) ...  
// but to be on the side of good programming put the return type*int power (int number, int pwr)  
{  
 int result = 1;  
 int i;  
 for (i=0; i<pwr; i++)  
 result = result \* number;  
 return result;  
}  
  
void message()  
{  
 printf("\n|=====================void message()===============================|\n");  
 printf( "|this function have the parameter list like ... () |\n");  
 printf( "|but it can be called like message(1.0). why? |\n");  
 printf( "|in C empty parameter list means parameter checking is disabled. |\n");  
 printf( "|So, whether it is message() or message(1.0) does not matter ! |\n");  
 printf( "|==================================================================|\n");  
}  
  
void message2(void)  
{  
 printf("\n|=====================void message2(void)==========================|\n");  
 printf( "|this function have the parameter list like ... (void) |\n");  
 printf( "|it can be called like message2(1.0). |\n");  
 printf( "|in C void in parameter list means function takes no argument |\n");  
 printf( "|message2(1.0) results in \"too many arguments to function\" error |\n");  
 printf( "|==================================================================|\n");  
  
}  
  
*//NOTES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
  
/\*  
"For the functions we write ourselves, we use the term 'definition.'  
 The general form of a function definition in C is as follows:  
  
[Return type] <function name> ([parameters]) { ... ... }"  
\*/  
  
/\*  
  
Return Values of Functions  
  
The value that a function sends back to the calling function at the end of its execution is called the function's return value.  
Not every function is required to produce a return value. The return values of functions can be used for different purposes:  
  
 1 - Some functions are designed to obtain a single value. They return the obtained value to the calling function as the return value. For example:  
  
y = pow(2, 3);  
  
The pow function is a standard C function. It calculates the power of the number specified by the first parameter raised to the power specified by the second parameter, and then sends the calculated value back to the calling function as the return value. In the above example, the third power of 2 is calculated by this function and the result is assigned to the variable y.  
  
 2 - Some functions' return values provide information about the success of the operations performed during the function's execution. In other words, the return values of these functions are used for testing purposes. Return values explain whether the desired operation was successful or not. For example:  
  
p = malloc(200);  
  
In this case, the programmer wants to allocate a block of memory 200 bytes long. However, they must also test whether this operation was successful. Immediately after the malloc call, the value of the p variable will be checked to determine the success of the operation. Therefore, the return value of the malloc function indicates whether the operation has been successfully completed.  
  
 3 - Some functions test the arguments passed to them based on certain criteria. The return value they produce indicates the result of the test. For example:  
  
if (isspace(ch)) {  
 ...  
}  
Here, the isspace function tests whether the character passed as an argument is a whitespace character. If it is a whitespace character, isspace will return a non-zero value, and if it is not, it will return 0. Based on the return value, different operations can be performed in the calling function.  
  
 4 - Some functions perform a specific task and, in addition, produce a return value that completes their purpose. For example:  
  
x = printf("Hello World\n");  
In this case, the printf function is used to print "Hello World" to the screen. However, it also returns the number of characters printed as a return value.  
  
 5 - Sometimes return values are not needed. For instance, a function designed solely to clear the screen does not need a return value.  
  
clrscr();  
The clrscr function only clears the screen, so it does not need a return value.  
  
There are also types for function return values. The return values of functions can be of any type, and their types are specified when the function is declared.  
\*/*

# *//* functions\_io.c

*// this file is about character input output functions*

#include <stdio.h>  
#include <conio.h>  
void functions\_io()  
{  
 printf("\n-----functions\_io\n");  
 printf("Functions to receive character input from the keyboard, other than scanf()\n");  
 char ch;  
  
 printf("\nenter an alphanumeric character using keyboard {a-z, A-Z, 0-9}:\n");  
 ch = getchar(); *//int getchar(void)  
 // there is a need to press enter. the character is displayed upon entering.* printf("...character is %c and the ASCII number is %d\n", ch, ch);  
 *// enter an alphanumeric character using keyboard {a-z, A-Z, 0-9}:  
 // p  
 // ...character is p and the ASCII number is 112* printf("\nenter an alphanumeric character using keyboard {a-z, A-Z, 0-9}:\n");  
 ch = getch(); *//int getch(void)  
 //(getch() is a non-standard func. which is present in the file of conio.h. !)  
 // no need to press enter, the character is not displayed upon entering.* printf("...character is %c and the ASCII number is %d\n", ch, ch);  
 *// enter an alphanumeric character using keyboard {a-z, A-Z, 0-9}:  
 // ...character is p and the ASCII number is 112* printf("\nenter an alphanumeric character using keyboard {a-z, A-Z, 0-9}:\n");  
 ch = getche(); *//int getche(void)  
 //(getche() is a non-standard func. which is present in the file of conio.h. !)  
 // no need to press enter, the character is displayed upon entering.* printf("...character is %c and the ASCII number is %d\n", ch, ch);  
 *// enter an alphanumeric character using keyboard {a-z, A-Z, 0-9}:  
 // p...character is p and the ASCII number is 112* printf("Functions to display characters as output on the screen, other than printf() \n");  
  
 putchar(ch); *//int putchar(int ch)* putch(ch); *//void putch(int ch)  
  
 // putchar(): This function is used to print one character on the screen,  
 // and this may be any character from C characterset  
 // (i.e it may be printable or non printable characters).  
 // putchar() return a value.  
  
 // putch(): The putch() function is used to display all alphanumeric characters  
 // throught the standard output device like monitor.  
 // this function display single character at a time.  
 // putch() does not return a value.  
 // Non-Standard Function, putch() is not part of the C standard library.*}  
*//NOTES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
  
/\*  
getchar: Requires pressing the Enter key; the character received is visible on the screen.  
getch: Does not require pressing the Enter key; the character received is not visible on the screen.  
getche: Does not require pressing the Enter key; the character received is visible on the screen.  
\*/*

// printf\_scanf.c  
*// this file is about printing different type of values, associated with format specifiers*void printf\_scanf()  
{  
 printf("\n-----printf\_scanf\n");  
 int x;  
 float y;  
  
 x = 125;  
 y = 200.0f;  
 printf("x = %d\ny = %f\n", x, y);  
 *// format specifier %d is matched with x  
 // format specifier %f is matched with y  
  
 //x = 125  
 //y = 200.000000  
  
 //When printing a short number,  
 //we use the h character before the d, o, u, or x characters  
 //{hd, hu, ho,hx}.* short int n1 = -32768; *// the number is SHRT\_MIN* printf("%hd\n", n1);  
 printf("%d\n", n1); *// sizeof(int) >= sizeof(short int), correct use* printf("%i\n\n", n1); *// sizeof(int) >= sizeof(short int), correct use  
 //-32768  
 //-32768  
 //-32768* unsigned short int n2 = 65535; *// the number is USHRT\_MAX* printf("%hu\n", n2);  
 printf("%ho\n", n2);  
 printf("%hx\n\n", n2);  
 *//65535  
 //177777  
 //ffff* int n3 = -2147483648; *// the number is INT\_MIN* printf("%d\n", n3);  
 printf("%i\n\n", n3);  
 *//-2147483648  
 //-2147483648* unsigned int n4 = 4294967295; *// the number is UINT\_MAX* printf("%u\n", n4);  
 printf("%o\n", n4);  
 printf("%x\n\n", n4);  
 *//4294967295  
 //37777777777  
 //ffffffff  
  
 //When printing a long number,  
 //we use the h character before the d, o, u, or x characters  
 //{ld, lu, lo,lx}.* long int n5 = -2147483648 ; *// the number is LONG\_MIN  
 // It can be -9223372036854775808 as -2^63 if sizeof(long int)==8  
 // but for my computer it is 4, -2147483648 is the LONG\_MIN* printf("%ld\n", n5);  
 printf("%d\n", n5); *// use only when sizeof(int) == sizeof(long int), correct use in this computer.* printf("%i\n\n", n5); *// use only when sizeof(int) == sizeof(long int), correct use in this computer.  
 //-2147483648  
 //-2147483648  
 //-2147483648* unsigned long int n6 = 4294967295; *// the number is ULONG\_MAX  
 // It can be 18446744073709551615 as 2^64-1 if sizeof(long int) == 8  
 // but for my computer it is 4, 4294967295 is the ULONG\_MAX.* printf("%lu\n", n6);  
 printf("%lo\n", n6);  
 printf("%lx\n\n", n6);  
  
 *// for n6 = 4294967296 (ULONG\_MAX+1), printf() displays 0.  
 //4294967295  
 //37777777777  
 //ffffffff* printf("\nfor the following codes compiler give incorrect output when used for this computer\n");  
 printf("for this computer long int holds 4 bytes, not 8 bytes. that is the reason\n");  
  
 printf("%d %d %d\n", sizeof(short int), sizeof(int), sizeof(long int));  
 long int n7 = -9223372036854775808 ;  
 *// if we assume long int hold 8 bytes !! (it does not)  
 // the number is supposedly LONG\_MIN for 8 byte long int* printf("%ld\n", n7); *// 0, {not correct, the number is not in the range}  
 // %d and %i cannot be used when sizeof(int) != sizeof(long int)* printf("%d\n", n7); *// 0 , {not correct, the number is not in the range}* printf("%i\n\n", n7); *// 0, {not correct, the number is not in the range}* unsigned long int n8 = 18446744073709551615;  
 *// if we assume long int hold 8 bytes !! (it does not)  
 // the number is supposedly ULONG\_MAX for 8 byte long int* printf("%lu\n", n8); *// 4294967295  
 //{not correct, the number is not in the range}* printf("%lo\n", n8); *// 37777777777  
 //{not correct, the number is not in the range}* printf("%lx\n\n", n8); *// ffffffff  
 //{not correct, the number is not in the range}  
  
 // now check printf\_scanf2.c*}  
  
  
*//NOTES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
  
/\*  
printf is actually a function with very detailed features.  
printf writes the characters within quotation marks to the screen.  
However, it does not print the % characters it encounters within the quotation marks.  
The printf function interprets the % characters along with the character next to them as format specifiers.  
The format specifiers are matched one-to-one with the parameters written after the quotation marks  
  
\*/  
  
/\*  
%d Writes int type in decimal system.  
%i Writes int type in decimal system.  
%ld Writes long type in decimal system.  
%hd Writes short type in decimal system.  
%x Writes unsigned int type in hexadecimal system.  
%X Writes unsigned int type in hexadecimal system (symbols in uppercase).  
%lx Writes unsigned long type in hexadecimal system.  
%u Writes unsigned int type in decimal system.  
%o Writes unsigned int type in octal system.  
%f Writes float and double types in decimal system.  
%lf Writes double type in decimal system.  
%e Writes real numbers in exponential format.  
%c Prints char or int type as a character representation.  
%s Prints as a string.  
%Lf Writes long double type in decimal system.  
\*/*

# *//* printf\_scanf2.c

*// this file is about printing long int type values when long int actually holds 8 byte information.*#include <stdio.h>  
  
void printf\_scanf2()  
{  
 printf("\n-----printf\_scanf2\n");  
  
  
 *// \*\*\*use https://www.tutorialspoint.com/compile\_c\_online.php\*\*\*\*//  
 // \*\*\*to compile remaining codes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
 // \*\*\*the int holds 4 bytes, long int holds 8 bytes\*\*\*\*\*\*\*\*\*\*\*\*\*\*//* printf("%d %d %d\n", sizeof(short int), sizeof(int), sizeof(long int));  
 *// 2 4 8* long int n7 = -9223372036854775808 ; *// the number is LONG\_MIN  
 // %d and %i cannot be used when sizeof(int) != sizeof(long int)* printf("%ld\n", n7); *//-9223372036854775808* printf("%d\n", n7); *// does not display the correct value* printf("%i\n\n", n7); *// does not display the correct value* unsigned long int n8 = 18446744073709551615;  
 printf("%lu\n", n8); *//18446744073709551615* printf("%lo\n", n8); *//1777777777777777777777* printf("%lx\n\n", n8); *//ffffffffffffffff*}

# *//* printf\_scanf3.c

*// this file is about the use of scanf() function*#include <stdio.h>  
  
void printf\_scanf3()  
{  
 printf("\n-----printf\_scanf3\n");  
 int x;  
 double y;  
  
 printf("\nplease enter values for x (int type) and y ( double type):\n");  
 printf("<use space, tab and enter keys: i.e type 12 23.6 and press enter>\n" );  
 *//12<space>23.6<enter>  
  
 //12<tab>23.6<enter>  
  
 //12<enter>  
 //23.6 <enter>* scanf("%d%lf", &x, &y);  
  
 printf("\n x = %d, y = %lf\n", x, y);  
 printf("\nthe size of int variable = %d and the size of double variable = %d\n", sizeof(x), sizeof(y));  
}  
  
  
*//NOTES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
  
/\*  
The first parameter of the scanf function is also a string. However, this string contains format specifiers related to the information to be received from the keyboard. These format specifiers are predefined and appear alongside the % character.  
  
We can say that the format specifiers used by the scanf function are the same as those used in the printf function. The only significant difference is in the format specifiers for real numbers. While the printf function can print both float and double data types to the screen using the %f format, the scanf function uses the %f format specifier only for float data types. For double types, the scanf function uses the %lf format specifier.  
  
In the format section of the scanf function, there should be nothing other than format specifiers. While the printf function would print characters other than format specifiers within the quotation marks to the screen, the scanf function does not print characters other than format specifiers in the string, as these characters would carry a completely different meaning. Therefore, do not include anything other than format specifiers in this section without understanding how the function works, as even a space can have a different meaning.  
\*/*

# *//* if\_statements.c

*// this file is about the use of conditional statements and character test functions in C  
// check https://tr.wikipedia.org/wiki/ASCII for ASCII characters*

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

int \_is\_upper(char ch);  
*// isupper, islower, isdigit, isalpha is the standard C versions of these functions with the same use.  
// these are sample functions we typed here*int \_is\_lower(char ch);  
int \_is\_digit(char ch);  
int \_is\_alpha(char ch);  
  
void if\_statements()  
{  
 printf("\n-----if\_statements\n");  
  
 int x = 1;  
 int m;  
  
 m = (x > 5) ? 1 : 0;  
 *// ? : conditional operator* printf("m = %d\n", m);  
  
 *// same as* if (x>5)  
 m = 1;  
 else  
 m = 0;  
  
 printf("m = %d\n\n", m);  
  
 char ch = 'Q';  
  
 if (ch >= 'a' && ch <= 'z')  
 printf("character %c is a lowercase latter\n", ch);  
 else  
 printf("character %c is not a lowercase latter\n", ch);  
  
 ch = 'a';  
 if (ch >= 'a' && ch <= 'z')  
 printf("character %c is a lowercase latter\n", ch);  
 else  
 printf("character %c is not a lowercase latter\n", ch);  
  
 if (10)  
 printf("always true\n"); *// True Statement is executed* if (0) *//zero value lead us to the false statements of the if* ;  
 else  
 printf("always false\n"); *// False Statement is executed  
  
 // bool flag = false; bool is a C++ type, not in C.  
 // Thus, we cannot input exact boolean values for if conditions* int flag = 0;  
 if (flag != 0 )  
 printf ("flag is not zero\n");  
 else if (flag == 0 ) *// just else will be enough* printf ("flag is zero\n");  
  
 flag = 5;  
 if (flag)  
 printf ("flag is different than zero\n");  
 *// numbers other than zero lead us to the true statements of the if* flag = -5;  
 if (flag)  
 printf ("flag is different than zero\n");  
 *// numbers other than zero lead us to the true statements of the if  
  
 //if -5, Not true. Always use parentheses for conditions* if (-5) *// numbers other than zero lead us to the true statements of the if* printf("this is a true statement\n");  
  
 *//===================================================================  
 //is-functions  
 //===================================================================* char array1[6] = {'a', 'A', '1', '+', '@', '\0'};  
 *// the last one is just Null character to terminate any char array.* int i = 0;  
 for (i = 0; i<=4; i++)  
 {  
 if (\_is\_upper(array1[i]))  
 printf("the character %c is an uppercase letter\n", array1[i]);  
 else if (\_is\_lower(array1[i]))  
 printf("the character %c is a lowercase letter\n", array1[i]);  
 else if (\_is\_digit(array1[i]))  
 printf("the character %c is a digit\n", array1[i]);  
 else  
 printf("the character %c is not an alphanumeric char.\n", array1[i]);  
 }  
  
  
 *// isspace and isprint are standard C functions* printf("\n");  
 for (i = 0; i<=127; i++)  
 if (isprint(i)!=0)  
 printf("ASCII Code %d, the character %c is printable\n", i, i);  
  
 */\* space characters  
 ' ' space  
 '\n' newline  
 '\t' horizontal tab  
 '\v' vertical tab  
 '\f' form feed  
 '\r' Carriage return  
 \*/* printf("\n");  
  
 char array2[7] = {' ', '\n', '\t', '\v', '\f','\r', '\0'};  
 *// the last one is just Null character to terminate any char array.* for (i = 0; i<=5; i++)  
 if (isspace(array2[i]))  
 printf("ASCII Code %d, the character %c is a space character\n",array2[i], array2[i]);  
  
 *// only space character is printable  
 // thus we have a reasonable display only for space*}  
  
  
int \_is\_upper(char ch)  
{  
 if (ch >= 'A' && ch <= 'Z') *//[65...90]* return 1;  
 else  
 return 0;  
}  
  
int \_is\_lower(char ch)  
{  
 if (ch >= 'a' && ch <= 'a') *//[97...102]* return 1;  
 else  
 return 0;  
}  
  
int \_is\_alpha(char ch)  
*// checks if the character is an alphabetic character  
// a-z, A-Z*{  
 if ((ch >= 'a' && ch <= 'a') || (ch >= 'A' && ch <= 'Z') )  
 return 1;  
 else  
 return 0;  
}  
  
int \_is\_digit(char ch) *////[30..39]  
// checks if the character is a digit  
// a-z, A-Z*{  
 if (ch >= '0' && ch <= '9')  
 return 1;  
 else  
 return 0;  
}  
*//NOTES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

*/\* standard character test functions in C | #include <ctype.h> for these*

*isalpha: True if it is an alphabetic character, False otherwise.  
isupper: True if it is an uppercase letter, False otherwise.  
islower: True if it is a lowercase letter, False otherwise.  
isdigit: True if it is a numeric character, False otherwise.  
isxdigit: True if it is a character representing a hexadecimal number, False otherwise.  
isalnum: True if it is an alphabetic or numeric character, False otherwise.  
isspace: True if it is a whitespace character (space, carriage return, newline, vertical tab, form feed), False otherwise.  
ispunct: True if it is a punctuation character (any character that is not a control, alphanumeric, or whitespace character), False otherwise.  
isprint: True if it is a printable character (including space), False otherwise.  
isgraph: True if it is a visible character (excluding space), False otherwise.  
iscntrl: True if it is a control character or delete character (first 32 characters or character number 127), False otherwise.  
isascii: True if it is one of the first 128 characters in the standard ASCII table, False otherwise.  
  
\*/*

# // type\_conversions.c

*//this file is about automatic type conversions*

#include <stdio.h>double product(double num1, double num2);  
double addition (int param1, int param2);  
  
void type\_conversions()  
{  
 printf("\n-----type\_conversions\n");  
 *// There are four cases where an automatic type conversion will definitely occur:  
  
 // 1- If the operands of an arithmetic or logical expression are not of the same type:  
 // In this case, the type conversions performed are called general arithmetic type conversions.* int x=7, y=3;  
 float fl=0.45f;  
  
 if (fl \* x / y - 1)  
 printf("\nthese expression will not be zero\n\n"); *// this will be displayed  
 // (0.45f \* 7) -> 3.15f |  
 // (3.5f / 2 ) -> 1.575f  
 // (1.575f -1 ) -> 0.575f | this is not zero* else  
 printf ("\nthese expression will be zero\n\n");  
  
 *// 2 - When the assignment operator is used, if the right side of the assignment operator is not the same type as the left side:* double result;  
 long num1=15, num2=12;  
  
 result = num1 + num2;  
 printf("%lf\n", result);  
 *// The type conversions performed in this case are called assignment type conversions.  
  
 // 3 - When a function is called, if the type of an argument used is not the same as the corresponding parameter variable of the function:* result = product(num1, num2);  
 printf("%lf\n", result);  
  
 *//4 If there is a difference between the type of a return expression and the return type of the function:* printf("%lf\n", product( num1, num2));  
}  
  
double product(double num1, double num2) *// if the values for num1 and num2 are int type they will be converted to double*{  
 return num1 \* num2;  
}  
  
double addition (int param1, int param2)  
{  
 return (param1 + param2);  
 *// param1 + param2 is int type, but function returns a double type value. There is an automatic conversion for this*}

*/\*  
operand 1 operand 2*

|  |  |  |
| --- | --- | --- |
| *long double* | *double, float, unsigned long, long, int, unsigned int* | *The 2nd operand will be converted to long double.* |
| *double* | *float, unsigned long, long, int, unsigned int* | *The 2nd operand will be converted to double.* |
| *float* | *unsigned long, long, int, unsigned int* | *The 2nd operand will be converted to float* |
| *unsigned long* | *long, int, unsigned int* | *The 2nd operand will be converted to unsigned long* |
| *long* | *int, unsigned int* | *The 2nd operand will be converted to long.* |
| *unsigned int* | *int* | *The 2nd operand will be converted to unsigned int.* |

# // switch\_case.c

*//this file is about switch case structures and random number generation*

#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
int dayofyear(int day, int month, int year);  
void randomdate(void);  
void dispdate(int day, int month, int year);  
  
void switch\_case()  
{  
 printf("\n-----switch\_case\n");  
 *//--------------------------  
 //test the function randomdate() that generates and displays a random date between 01.01.1923 and 31.12.2000* int n = 20;  
 srand(time(NULL));  
 while (n-- > 0) {  
 randomdate();  
 putchar('\n');  
 }  
 *//--------------------------  
 //test the function dayofyear() that determines which day of the year a given date (day, month, year) is* n = 2;  
 int day, month, year;  
  
 while (n-- > 0)  
 {  
 printf("\nEnter a date as day month year: ");  
 scanf("%d%d%d", &day, &month, &year);  
 printf("It is the %dth day of the year %d.\n", dayofyear(day, month, year), year);  
 }  
 *//--------------------------  
 //test the function dispdate() that displays a given date on the screen in English format (e.g., 15th Aug. 2000)* n = 2;  
 while (n-- > 0)  
 {  
 printf("\nEnter a date as day month year: ");  
 scanf("%d%d%d", &day, &month, &year);  
 dispdate(day, month, year);  
 putchar('\n');  
 }  
 return 0;  
}  
  
*// A function that generates and displays a random date between 01.01.1923 and 31.12.2000*void randomdate(void) *// assuming the input values are reasonable. there is no check of the correctness of the input !*{  
 int day, month, year;  
 int monthdays;  
  
 year = rand() % 78 + 1923;  
 month = rand() % 12 + 1;  
  
 switch (month) {  
 case 4:  
 case 6:  
 case 9:  
 case 11: monthdays = 30; break;  
 case 2: monthdays = isleap(year) ? 29 : 28; break;  
 default : monthdays = 31;  
 }  
 day = rand() % monthdays + 1;  
  
 printf("%02d - %02d - %d", day, month, year);  
}  
  
*//A function that determines which day of the year a given date (day, month, year) is*int dayofyear(int day, int month, int year) *// assuming the input values are reasonable. there is no check of the correctness of the input !*{  
 int yearday = day;  
  
 switch (month - 1) {  
 case 11: yearday += 30;  
 case 10: yearday += 31;  
 case 9 : yearday += 30;  
 case 8 : yearday += 31;  
 case 7 : yearday += 31;  
 case 6 : yearday += 30;  
 case 5 : yearday += 31;  
 case 4 : yearday += 30;  
 case 3 : yearday += 31;  
 case 2 : yearday += isleap(year) ? 29 : 28;  
 case 1 : yearday += 31;  
 }  
 return yearday;  
}  
  
*//A function that displays a given date on the screen in English format (e.g., 15th Aug. 2000)*void dispdate(int day, int month, int year) *// assuming the input values are reasonable. there is no check of the correctness of the input !*{  
 printf("%2d", day);  
  
 switch (day) {  
 case 1 :  
 case 21:  
 case 31: printf("st "); break;  
 case 2 :  
 case 22: printf("nd "); break;  
 case 3 :  
 case 23: printf("rd "); break;  
 default : printf("th ");  
 }  
  
 switch (month) {  
 case 1 : printf("Jan "); break;  
 case 2 : printf("Feb "); break;  
 case 3 : printf("Mar "); break;  
 case 4 : printf("Apr "); break;  
 case 5 : printf("May "); break;  
 case 6 : printf("Jun "); break;  
 case 7 : printf("Jul "); break;  
 case 8 : printf("Aug "); break;  
 case 9 : printf("Sep "); break;  
 case 10: printf("Oct "); break;  
 case 11: printf("Nov "); break;  
 case 12: printf("Dec ");  
 }  
 printf("%d", year);  
}  
  
int isleap(int year)  
{  
 if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0)  
 return year;  
 return 0;  
}