Lecture-2

Overview (1/2)

- Tokens
- Character
- Keywords
- Identifiers
- Literals
- Special Symbols
- String
- Boolean
- Format Specifier

Overview (2/2)

- User Input and Output
- Macro in C
- Header Files
- Constants
- Operators
- Signed and Unsigned Numbers
- ASCII Code
- User Input
    - Integer, float, character, string

Tokens

- A token is referred to as the smallest unit in the source code of a computer language such as C.
- The term token is borrowed from the theory of linguistics- Just as a certain piece of text in a language (like English) comprises words (collection of alphabets), digits, and punctuation symbols.
- A compiler breaks a C program into tokens and then proceeds ahead to the next stages used in the compilation process.

Tokens

- The first stage in the compilation process is the tokenizer.
  - The tokenizer divides the source code into individual tokens, identifying the token type, and passing tokens one at a time to the next stage of the compiler.

- The parser is the next stage in the compilation.
  - It is capable of understanding the language's grammar. identifies syntax errors and translates an error-free program into the machine language.

Tokens

- A C source code also comprises tokens of different types. The tokens in C are of the following types −
    - Character set
    - Keyword tokens
    - Literal tokens
    - Identifier tokens
    - Operator tokens
    - Special symbol tokens

Character

- The C language identifies a character set that comprises
  - English alphabets – upper and lowercase (A to Z, as well as a to z),
  - digits 0 to 9, and certain other symbols with a special meaning attached to them.
  - In C, certain combinations of characters also have a special meaning attached to them.
    - For example, \n is known as a newline character. Such combinations are called escape sequences.

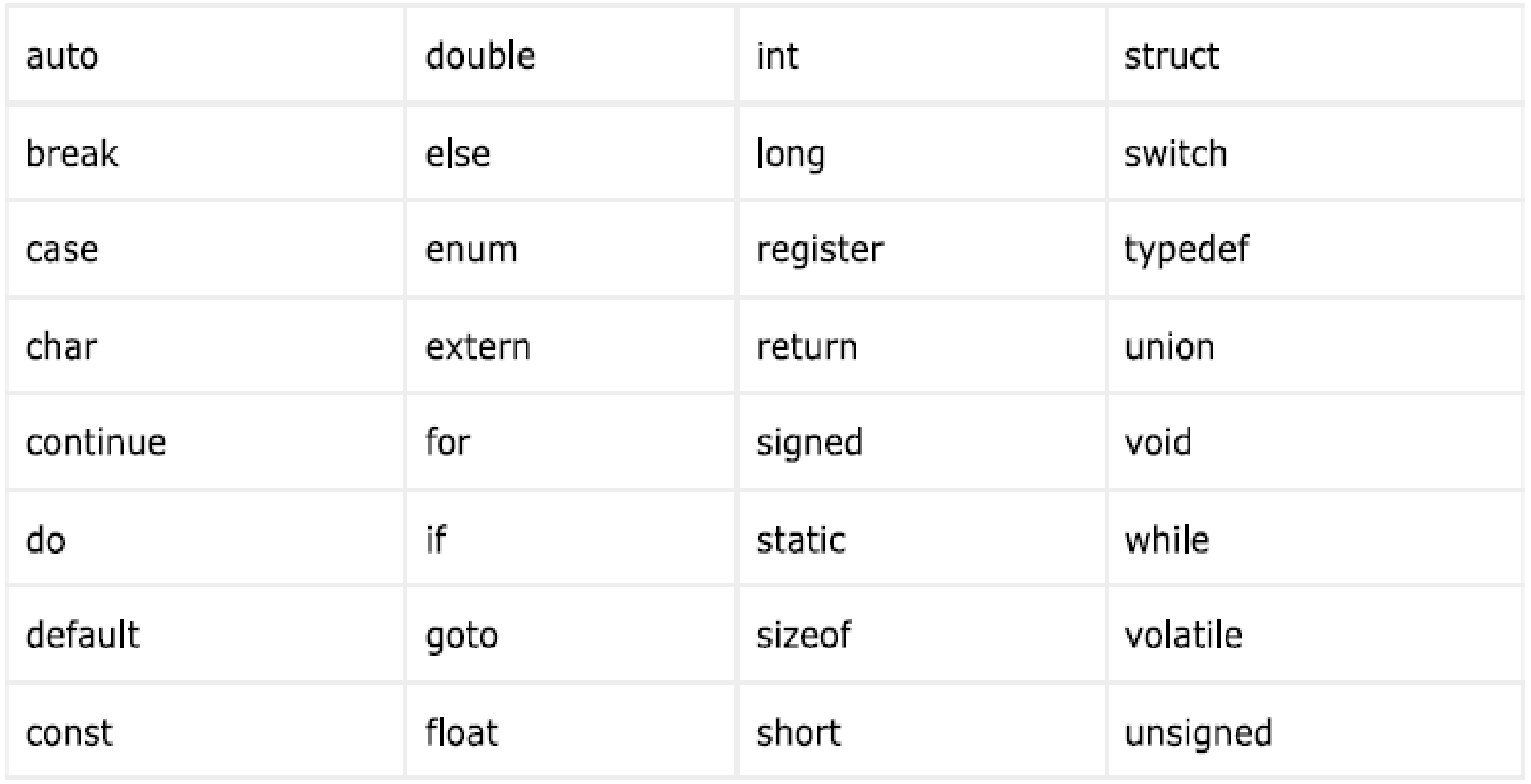
Character

- Here is the character set of C language −
  - Uppercase: A to Z
  - Lowercase: a to z
  - Digits: 0 to 9
  - Special characters: ! " # $ % & ' ( ) * + - . : , ; ` ~ = < > { } [ ] ^ _ \ /
  - A sequence of any of these characters inside a pair of double quote symbols ("") are used to represent a string literal.

Keywords

- In C, a predefined sequence of alphabets is called a keyword.
    - programming languages have fewer keywords.
    - To start with, C had 32 keywords, later on, few more were added in subsequent revisions of C standards.
    - All keywords are in lowercase. Each keyword has rules of usage (in programming it is called syntax) attached to it.
    - The C compiler checks whether a keyword has been used according to the syntax, and translates the source code into the object code.

Keywords



auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

Keywords

Primary Types C Keywords

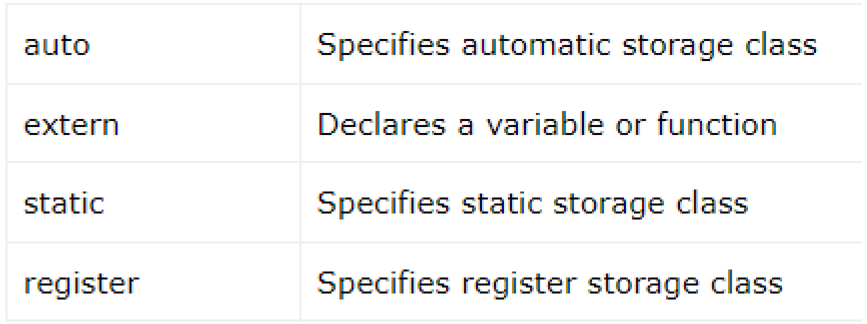
int	Declares an integer variable
long	Declares a long integer variable
short	Declares a short integer variable
signed	Declares a signed variable
double	Declares a double-precision variable
char	Declares a character variable
float	Declares a floating-point variable
unsigned	Declares an unsigned variable
void	Specifies a void return type

Storage Defined Types C Keywords

struct	Declares a structure type
typedef	Creates a new data type
union	Declares a union type
enum	Declares an enumeration type

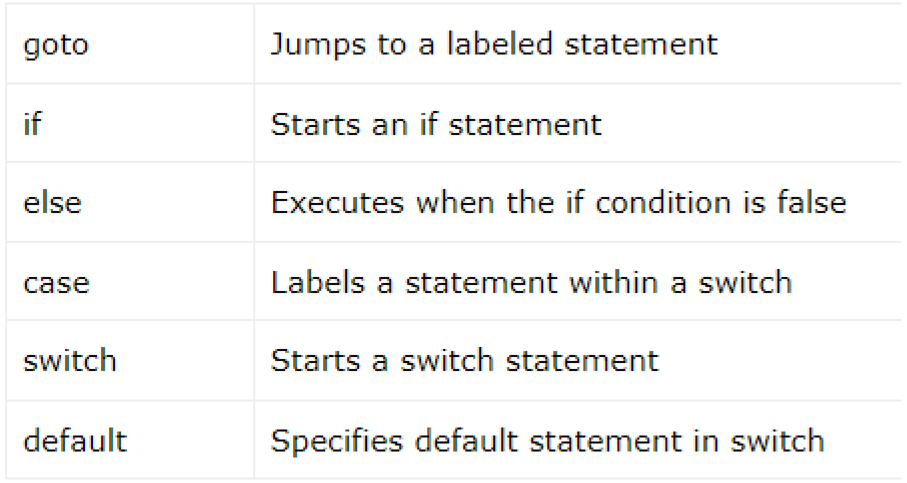
Keywords

Storage Types C Keywords



auto	Specifies automatic storage class
extern	Declares a variable or function
static	Specifies static storage class
register	Specifies register storage class

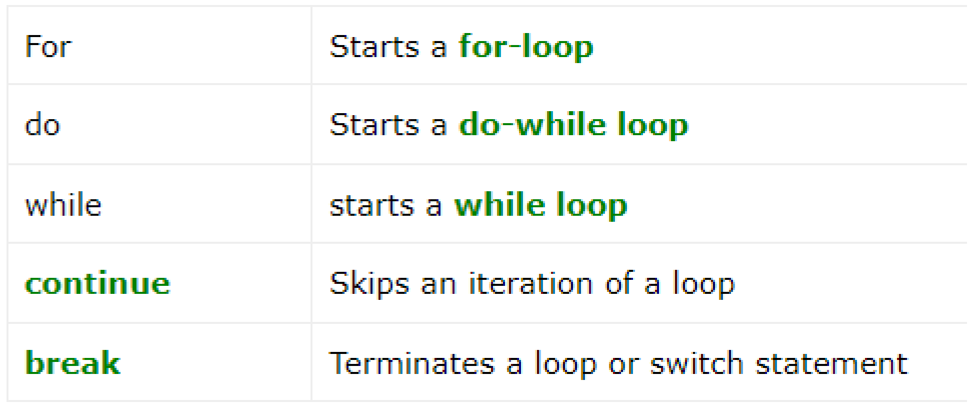
Conditional Keywords



goto	Jumps to a labeled statement
if	Starts an if statement
else	Executes when the if condition is false
case	Labels a statement within a switch
switch	Starts a switch statement
default	Specifies default statement in switch

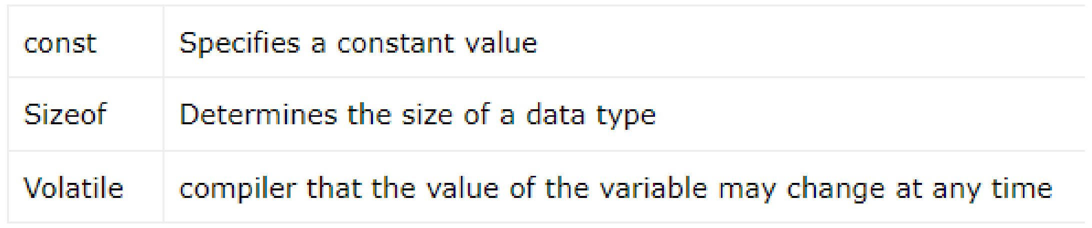
Keywords

Loops and Loops Control Keywords



For	Starts a **for-loop**
do	Starts a **do-while loop**
while	starts a **while loop**
continue	Skips an iteration of a loop
break	Terminates a loop or switch statement

Other C Keywords



const	Specifies a constant value
Sizeof	Determines the size of a data type
Volatile	compiler that the value of the variable may change at any time

Identifier

- Identifiers are the user-defined names given to make it easy to refer to the memory.
- It is also used to define various elements in the program, such as the function, user-defined type, labels, etc.
- When a variable or a function is defined with an identifier, the C compiler allocates it the memory and associates the memory location to the identifier.
- Example of Identifiers: Variable Identifier, function identifier, user defined type identifier, Typedef identifier, Label identifier, Enum identifier

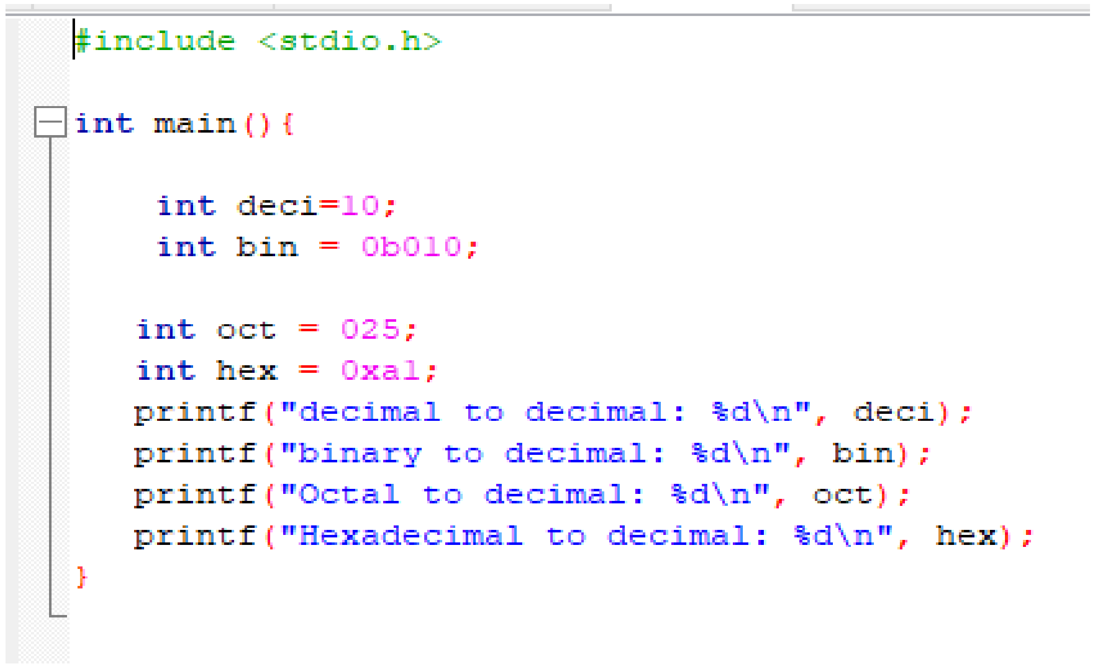
Literals

- The term "literal" in computer programming terminology refers to a textual representation of a value to be assigned to a variable.

- Example- Integer literal, floating point literals, character literals, string literals, escape sequence literals, etc.

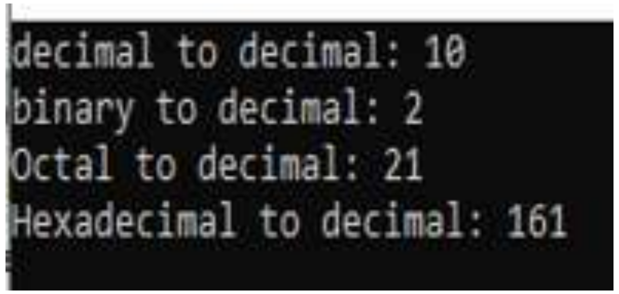
Integer Literals

Code



```c
#include <stdio.h>

int main(){

     int deci=10;
     int bin = 0b010;

    int oct = 025;
    int hex = 0xa1;
    printf("decimal to decimal: %d\n", deci);
    printf("binary to decimal: %d\n", bin);
    printf("Octal to decimal: %d\n", oct);
    printf("Hexadecimal to decimal: %d\n", hex);
}
```

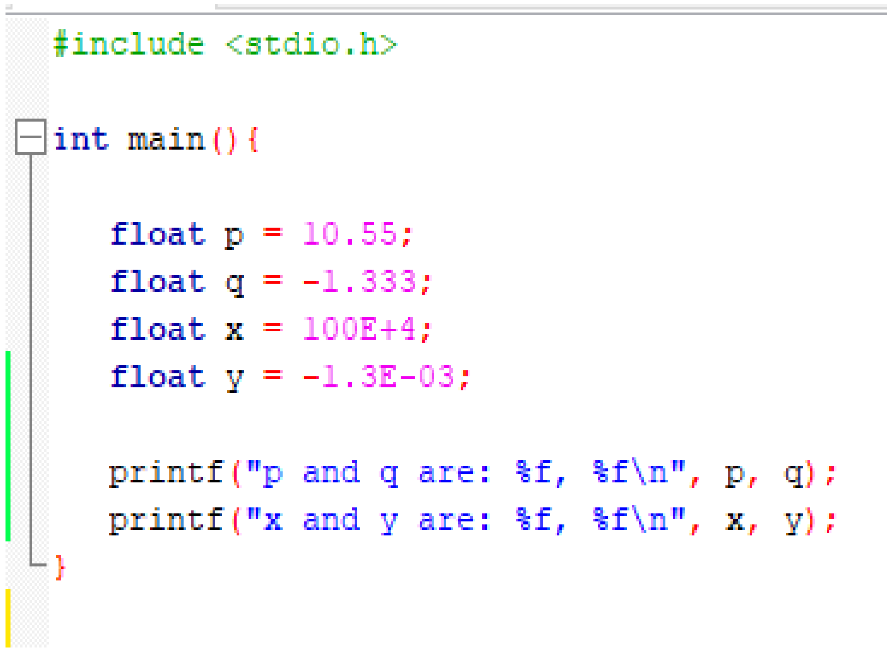
Output



```
decimal to decimal: 10
binary to decimal: 2
Octal to decimal: 21
Hexadecimal to decimal: 161
```

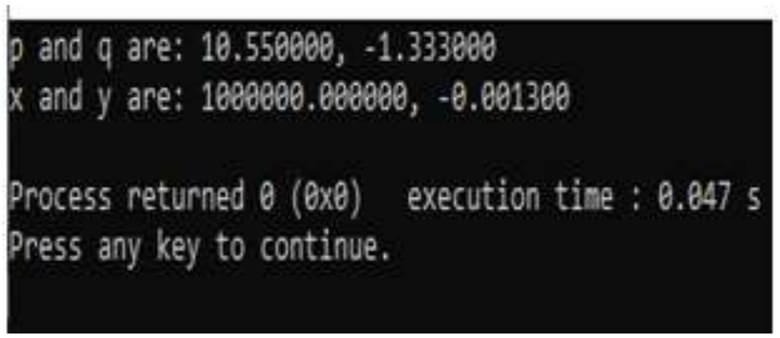
Floating Point Literals

Code



```
#include <stdio.h>

int main(){

    float p = 10.55;
    float q = -1.333;
    float x = 100E+4;
    float y = -1.3E-03;

    printf("p and q are: %f, %f\n", p, q);
    printf("x and y are: %f, %f\n", x, y);
}
```

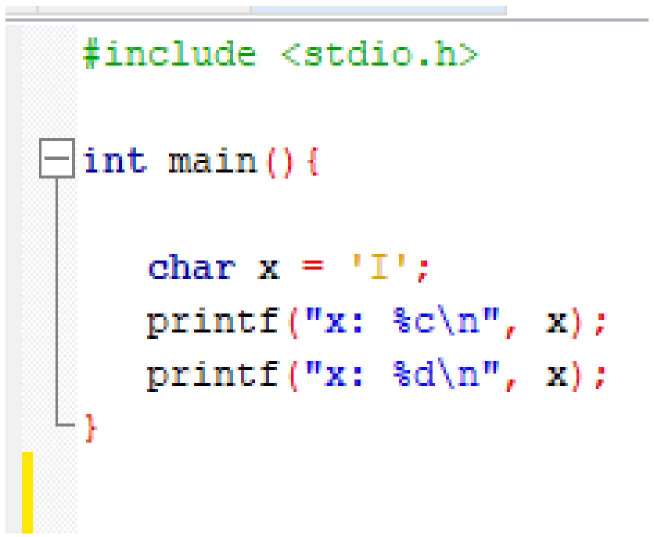
Output



```
p and q are: 10.550000, -1.333000
x and y are: 1000000.000000, -0.001300

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

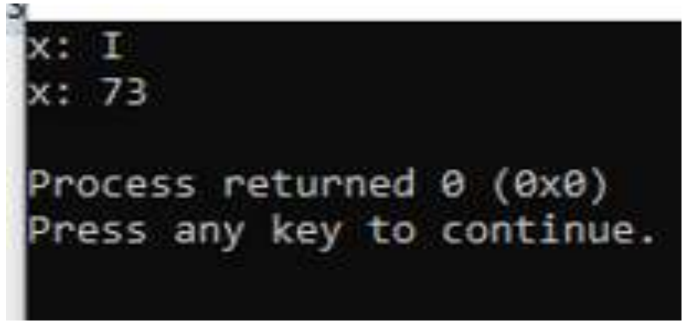
Character Literal

Code



```
#include <stdio.h>

int main(){

    char x = 'I';
    printf("x: %c\n", x);
    printf("x: %d\n", x);
}
```

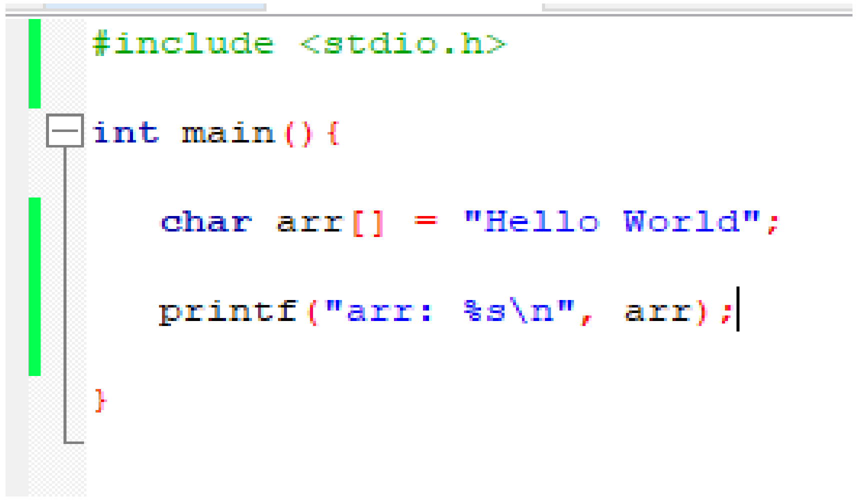
Output



```
x: I
x: 73

Process returned 0 (0x0)
Press any key to continue.
```

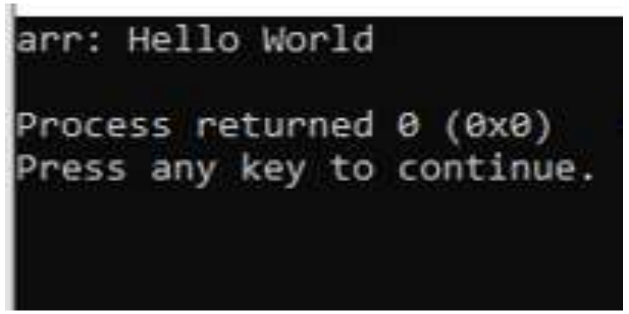
String Literal

Code



```
#include <stdio.h>

int main(){

    char arr[] = "Hello World";

    printf("arr: %s\n", arr);

}
```

Output

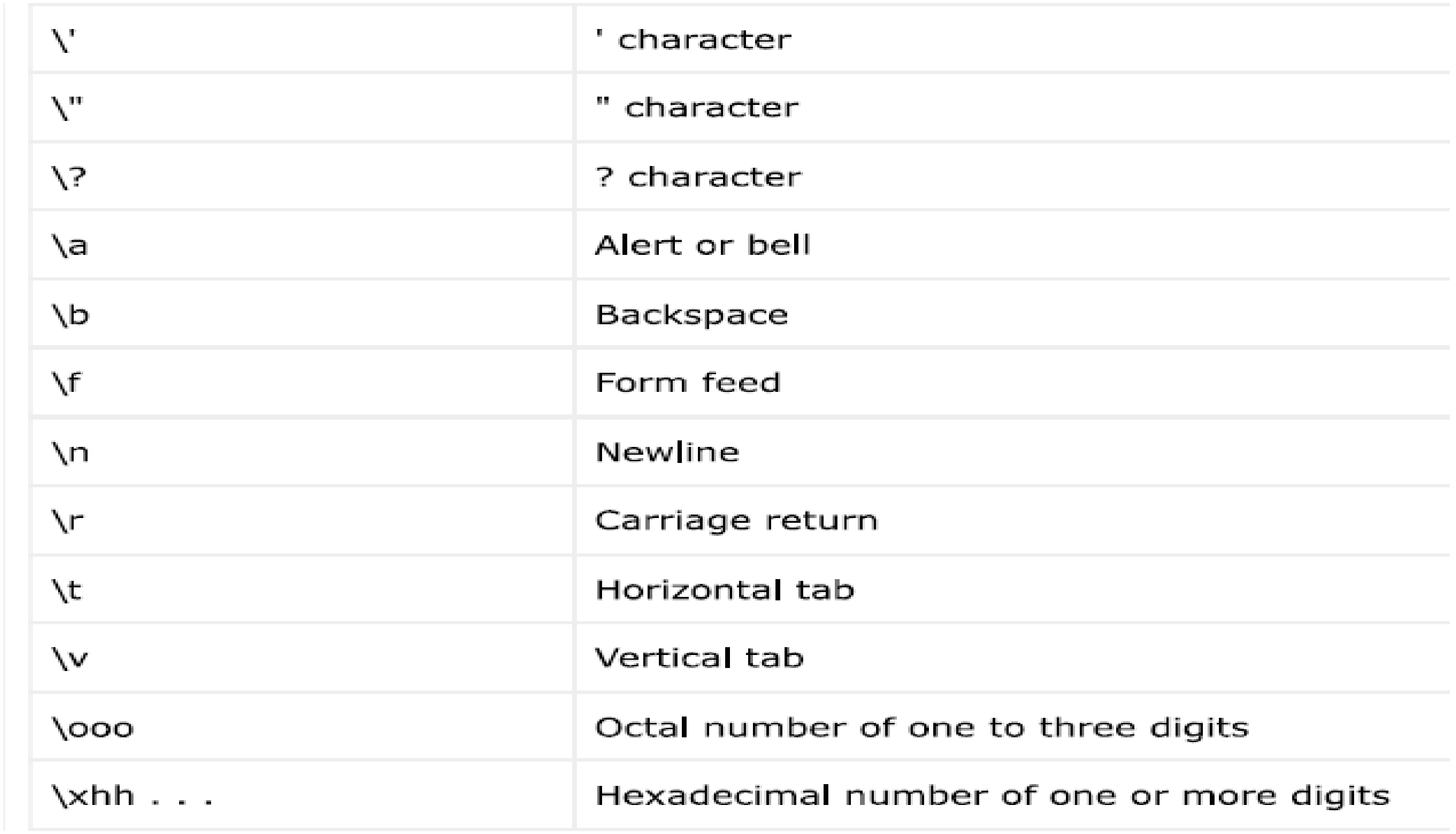


```
arr: Hello World

Process returned 0 (0x0)
Press any key to continue.
```

Escape Sequence

- C defines a number of escape sequences as a sequence of characters starting with "\" and an alternate meaning attached to the following characters.
- Even though an escape sequence consists of more than one characters, it is put inside single quotes.
- An escape sequence produces the effect of a single non-printable character.
  - For example, '\n' is an escape sequence that represents a newline character, with the same effect as pressing the Enter key.

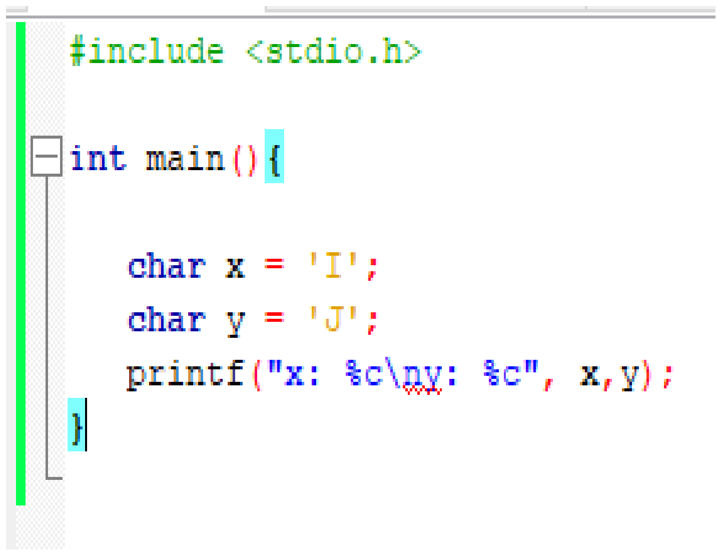
Escape Sequence



\'	' character
\"	" character
\?	? character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\ooo	Octal number of one to three digits
\xhh . . .	Hexadecimal number of one or more digits

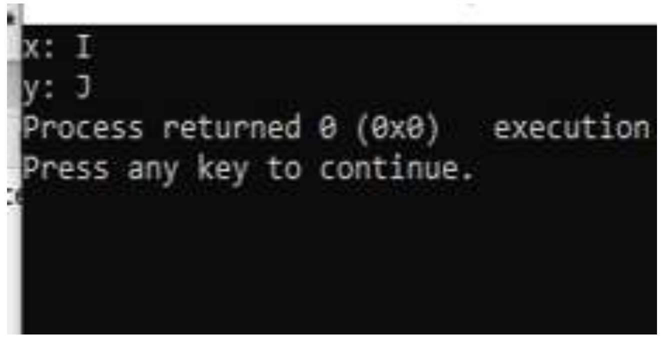
Escape Sequence Example (\n)

Code



```
#include <stdio.h>

int main(){

    char x = 'I';
    char y = 'J';
    printf("x: %c\ny: %c", x,y);
}
```

Output



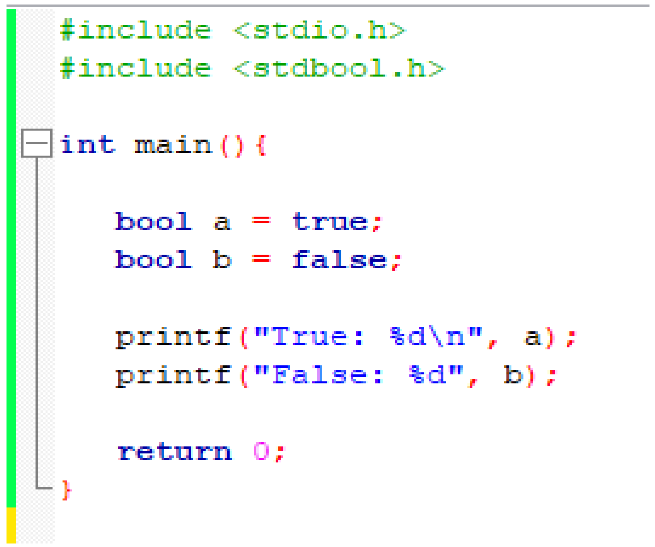
```
x: I
y: J
Process returned 0 (0x0)   execution
Press any key to continue.
```

Boolean

- In C programming, "ANSI C" refers to the standardized version of the C programming language defined by the American National Standards Institute (ANSI).
  - This standardization ensures that C code written according to the standard can be compiled and run on different systems with minimal changes.
  - Unlike the int, char or float types, the ANSI C standard doesn’t have a built-in or primary Boolean type.
- A Boolean or bool data generally refers to the one that can hold one of the two binary values: true or false (or yes/no, on/off, etc.).
- Even if the bool type is not available in C, the behaviour of Booleans can be implemented with the help of an enum type.
- The new versions of C compilers, complying with the C99 standard or later, support the bool type, which has been defined in the header file stdbool.h.

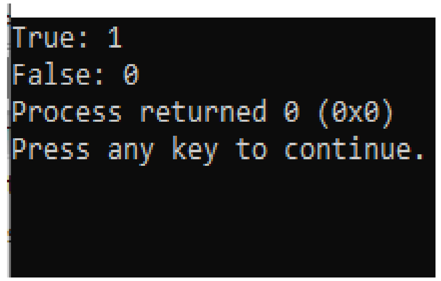
Boolean Example 1

Code



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

int main(){

    bool a = true;
    bool b = false;

    printf("True: %d\n", a);
    printf("False: %d", b);

    return 0;
}
```

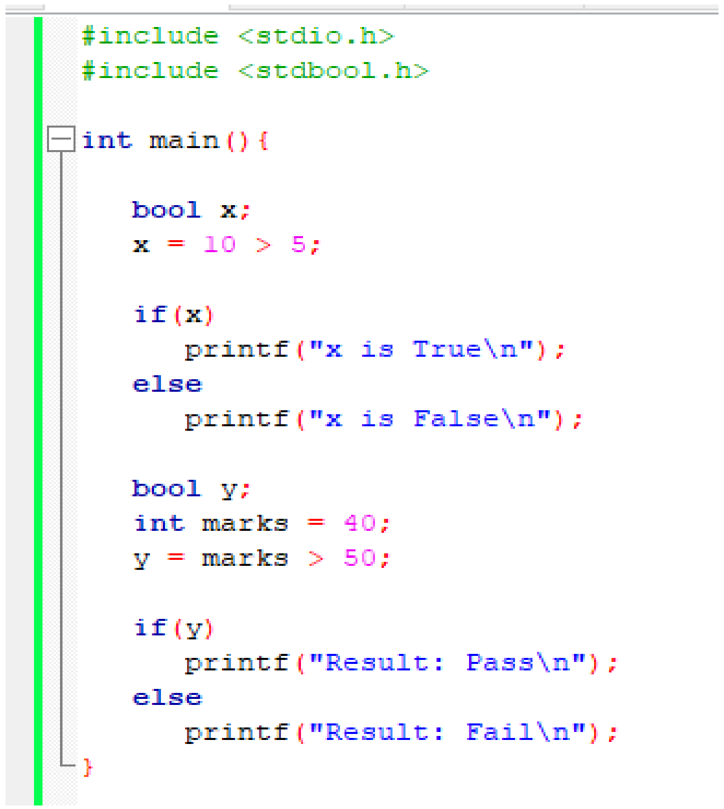
Output



```
True: 1
False: 0
Process returned 0 (0x0)
Press any key to continue.
```

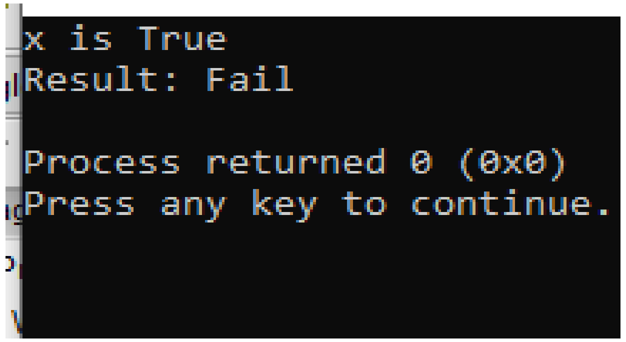
Boolean Example 2

Code



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

int main(){

    bool x;
    x = 10 > 5;

    if(x)
        printf("x is True\n");
    else
        printf("x is False\n");

    bool y;
    int marks = 40;
    y = marks > 50;

    if(y)
        printf("Result: Pass\n");
    else
        printf("Result: Fail\n");
}
```

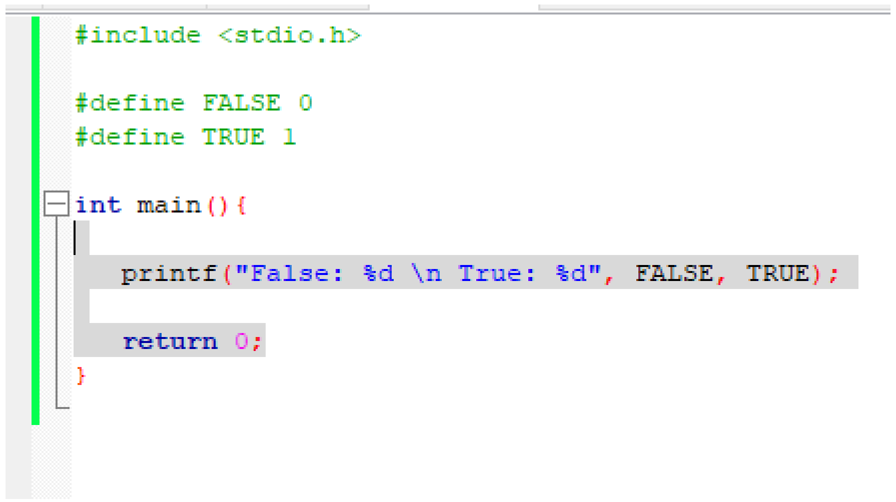
Output



```
x is True
Result: Fail

Process returned 0 (0x0)
Press any key to continue.
```

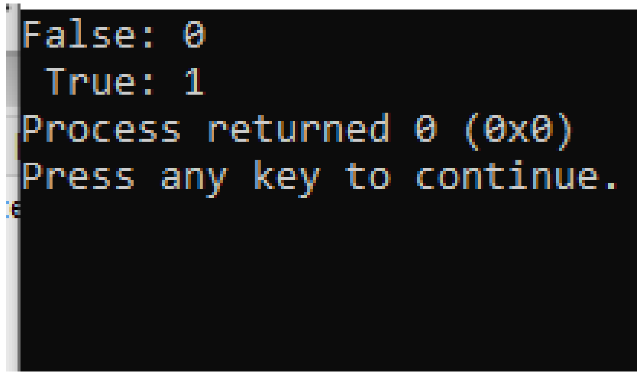
Boolean Example 3

Code



```c
#include <stdio.h>

#define FALSE 0
#define TRUE 1

int main(){

    printf("False: %d \n True: %d", FALSE, TRUE);

    return 0;
}
```

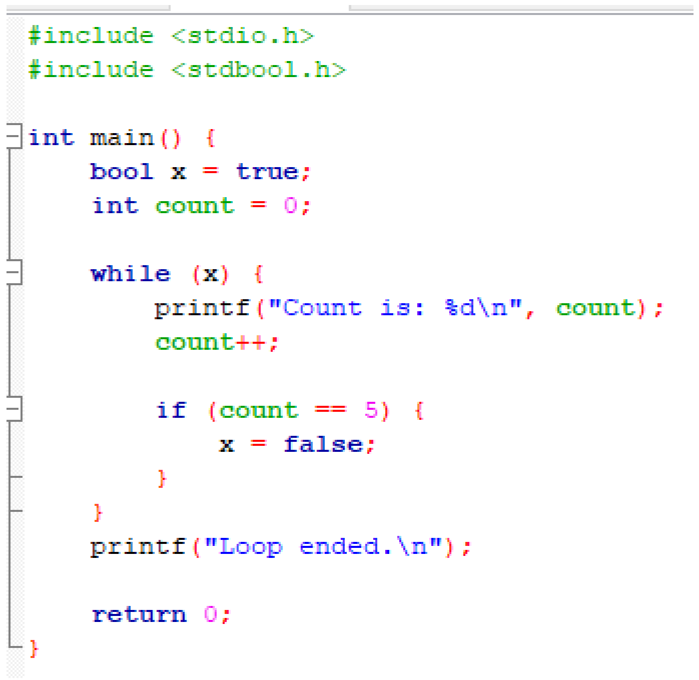
Output



```
False: 0
 True: 1
Process returned 0 (0x0)
Press any key to continue.
```

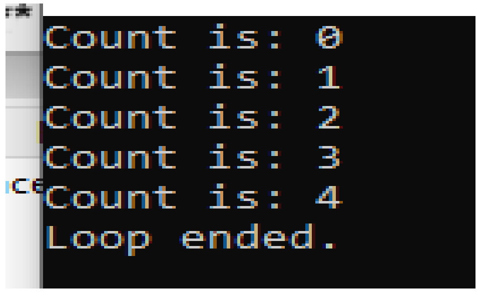
Boolean Example 4

Code



```c
#include <stdio.h>
#include <stdbool.h>

int main() {
    bool x = true;
    int count = 0;

    while (x) {
        printf("Count is: %d\n", count);
        count++;

        if (count == 5) {
            x = false;
        }
    }
    printf("Loop ended.\n");

    return 0;
}
```

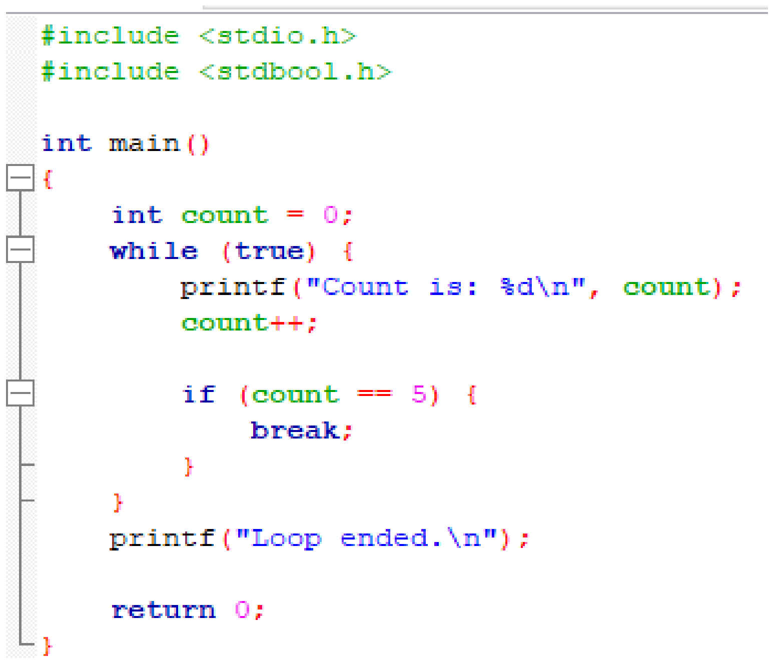
Output



```
Count is: 0
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Loop ended.
```

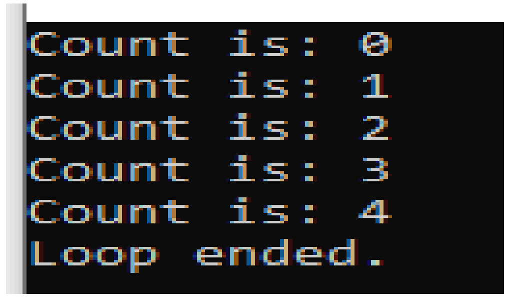
Boolean Example 5

Code



```c
#include <stdio.h>
#include <stdbool.h>

int main()
{
    int count = 0;
    while (true) {
        printf("Count is: %d\n", count);
        count++;

        if (count == 5) {
            break;
        }
    }
    printf("Loop ended.\n");

    return 0;
}
```

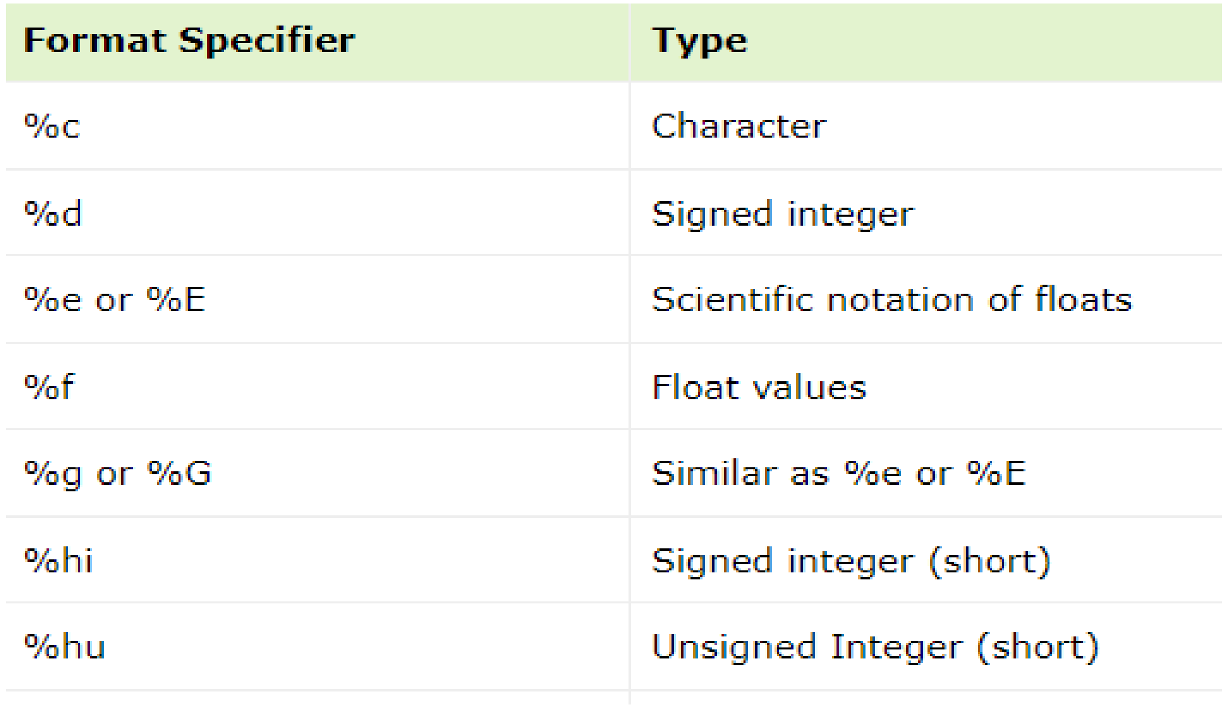
Output



```
Count is: 0
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Loop ended.
```

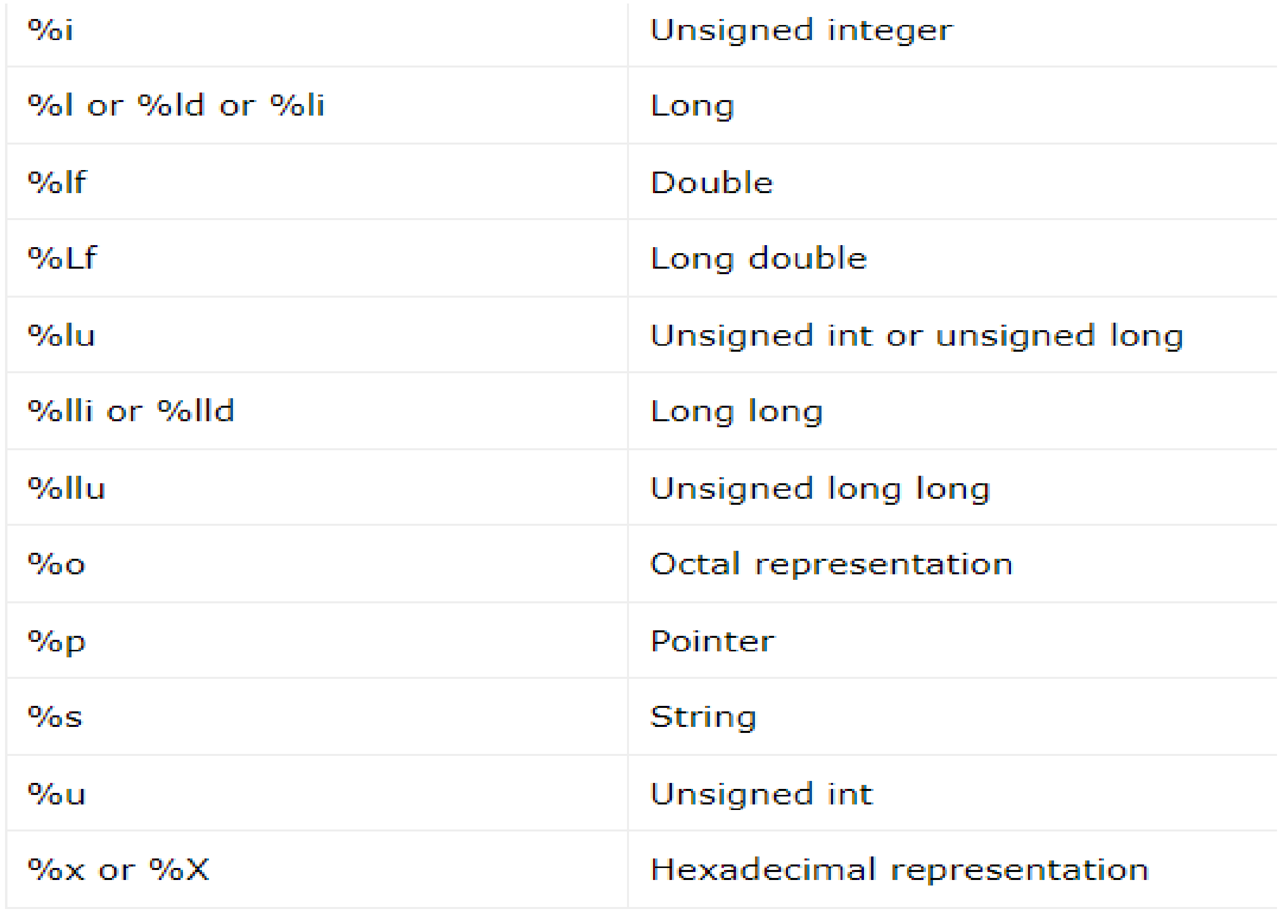
Format Specifier

- Format specifiers in C are certain special symbols used in the formatted console IO functions such as printf() and scanf(), as well as formatted file IO functions such as fprintf() and fscanf().



Format Specifier	Type
%c	Character
%d	Signed integer
%e or %E	Scientific notation of floats
%f	Float values
%g or %G	Similar as %e or %E
%hi	Signed integer (short)
%hu	Unsigned Integer (short)

Format Specifier



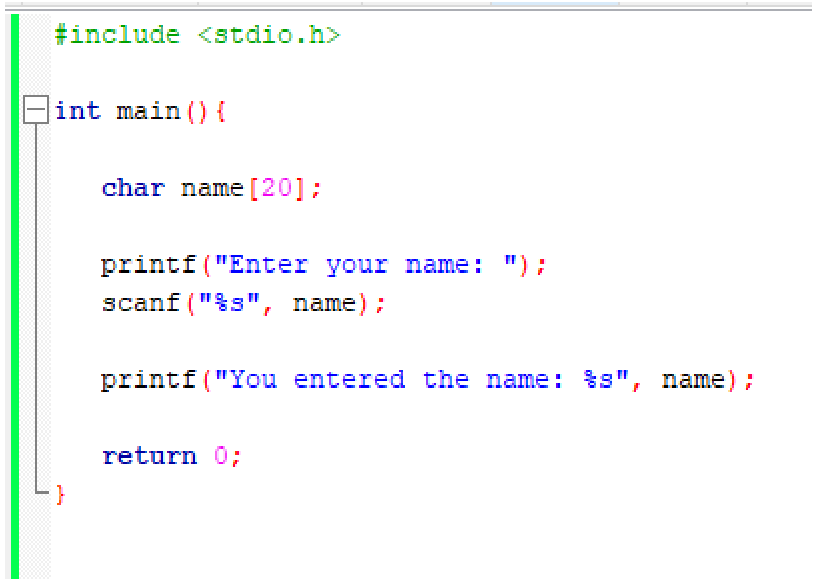
%i	Unsigned integer
%l or %ld or %li	Long
%lf	Double
%Lf	Long double
%lu	Unsigned int or unsigned long
%lli or %lld	Long long
%llu	Unsigned long long
%o	Octal representation
%p	Pointer
%s	String
%u	Unsigned int
%x or %X	Hexadecimal representation

User Input and Output Functions

- Input Functions:
  - scanf: Reads input from the keyboard.
  - gets: Reads a line of text from the keyboard.
  - fscanf: Reads input from a file.
  - fgets: Safely reads a string from a file.
  - getchar: Reads a single character from the keyboard.
- Output Functions:
  - printf: Writes output to the screen.
  - puts: Prints a string to the screen with a newline.
  - fprintf: Writes output to a file.
  - fputs: Writes a string to a file without adding a newline.
  - putchar: Outputs a single character to the screen.

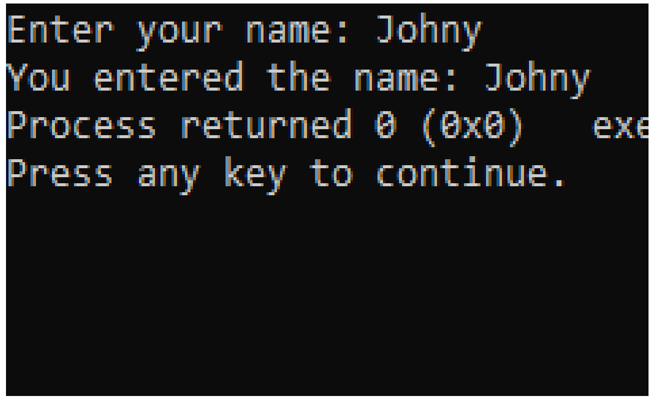
String Input Using scanf() Function

Code



```c
#include <stdio.h>

int main(){

    char name[20];

    printf("Enter your name: ");
    scanf("%s", name);

    printf("You entered the name: %s", name);

    return 0;
}
```

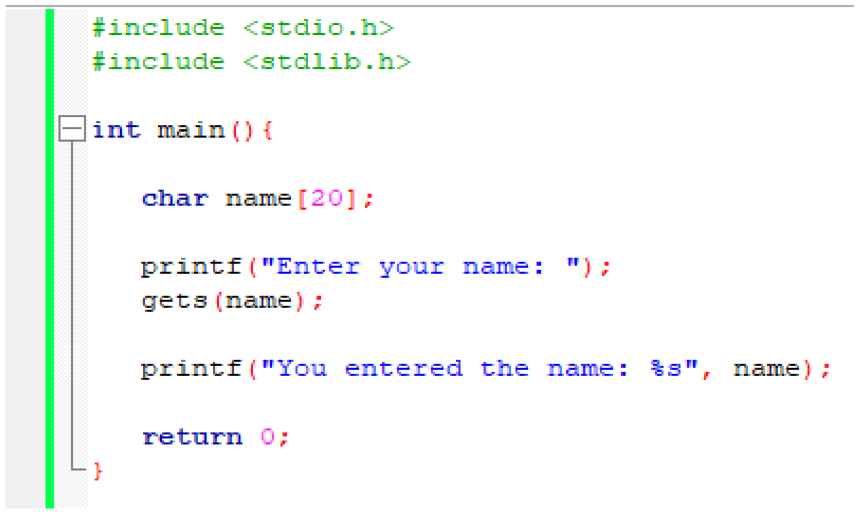
Output



```
Enter your name: Johny
You entered the name: Johny
Process returned 0 (0x0)   exe
Press any key to continue.
```

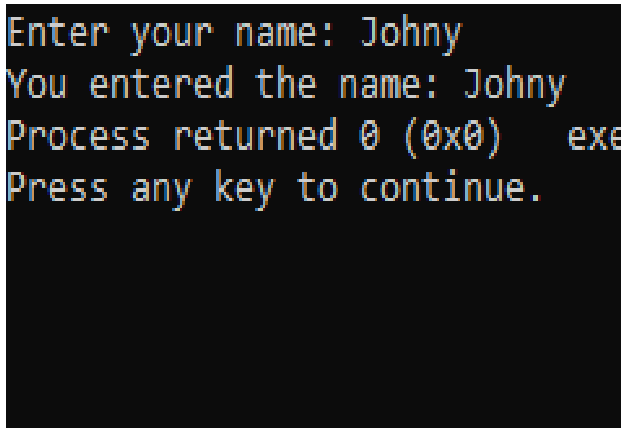
String Input Using gets() Function

Code



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

int main(){

    char name[20];

    printf("Enter your name: ");
    gets(name);

    printf("You entered the name: %s", name);

    return 0;
}
```

Output

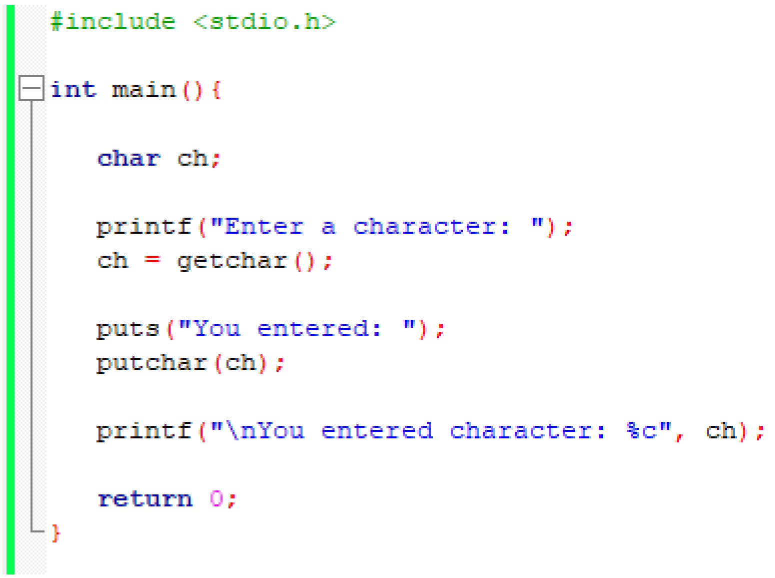


```
Enter your name: Johny
You entered the name: Johny
Process returned 0 (0x0)   exe
Press any key to continue.
```

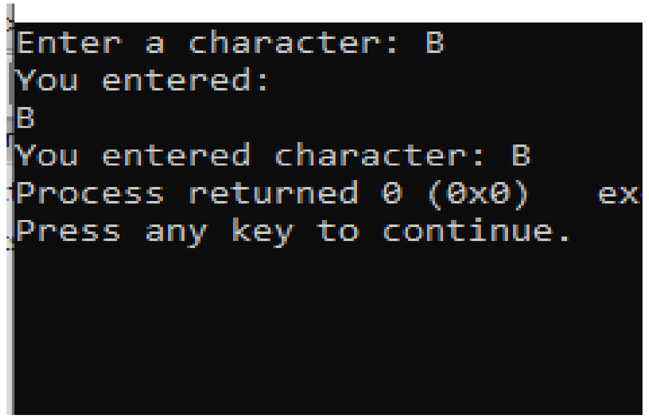
Single character Using getchar(), printing with puts() and putchar() function

Code

Output



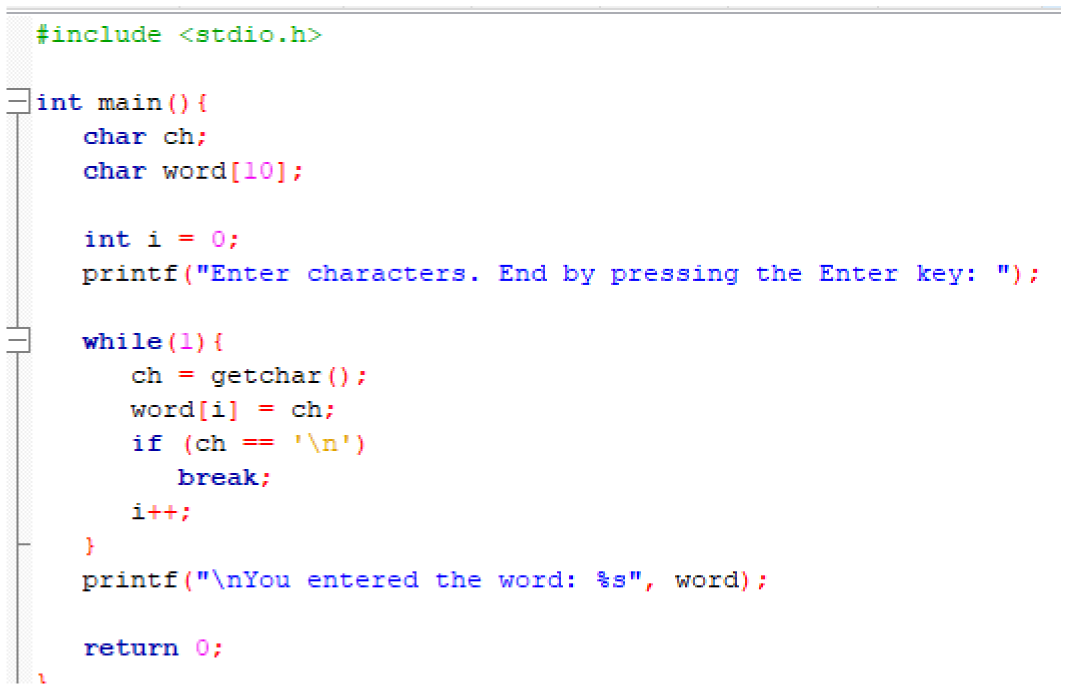
```
#include <stdio.h>

int main(){

    char ch;

    printf("Enter a character: ");
    ch = getchar();

    puts("You entered: ");
    putchar(ch);

    printf("\nYou entered character: %c", ch);

    return 0;
}
```



```
Enter a character: B
You entered:
B
You entered character: B
Process returned 0 (0x0)   ex
Press any key to continue.
```

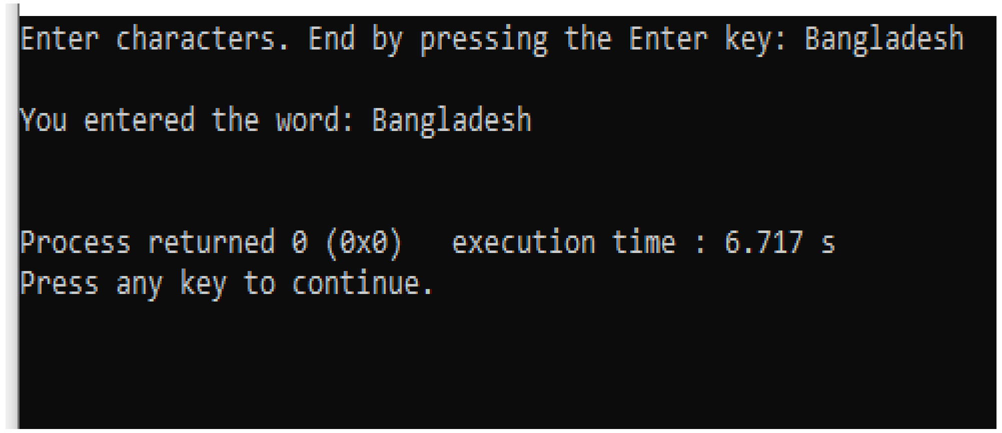
Sequence of characters Using getchar() Function

Code



```c
#include <stdio.h>

int main(){
    char ch;
    char word[10];

    int i = 0;
    printf("Enter characters. End by pressing the Enter key: ");

    while(1){
        ch = getchar();
        word[i] = ch;
        if (ch == '\n')
            break;
        i++;
    }
    printf("\nYou entered the word: %s", word);

    return 0;
}
```

Output



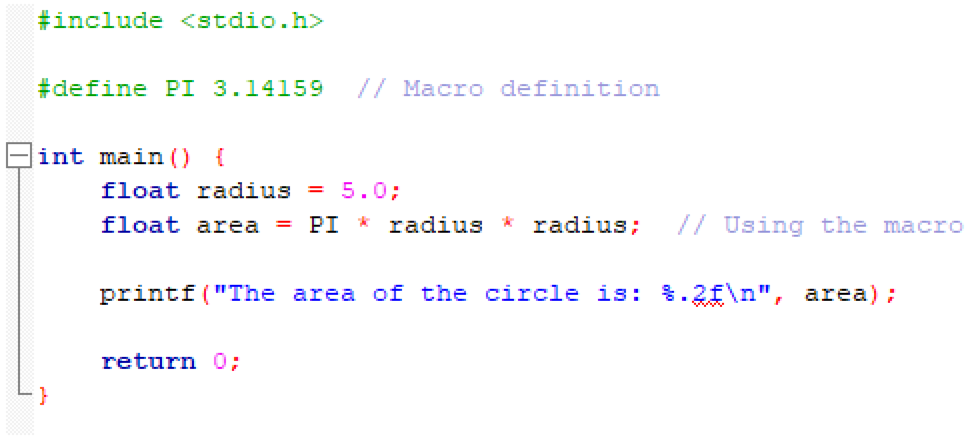
```
Enter characters. End by pressing the Enter key: Bangladesh

You entered the word: Bangladesh


Process returned 0 (0x0)   execution time : 6.717 s
Press any key to continue.
```

Macro in C Programming

- In C programming, a macro is a fragment of code that has been given a name.
- Whenever the name is used, it is replaced by the contents of the macro. Macros are defined using the #define directive.

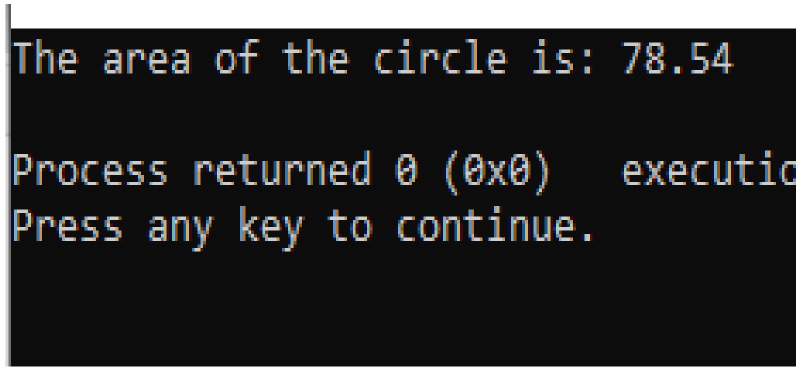
Example of Macro in C Programming

Code



```
#include <stdio.h>

#define PI 3.14159  // Macro definition

int main() {
    float radius = 5.0;
    float area = PI * radius * radius;  // Using the macro

    printf("The area of the circle is: %.2f\n", area);

    return 0;
}
```

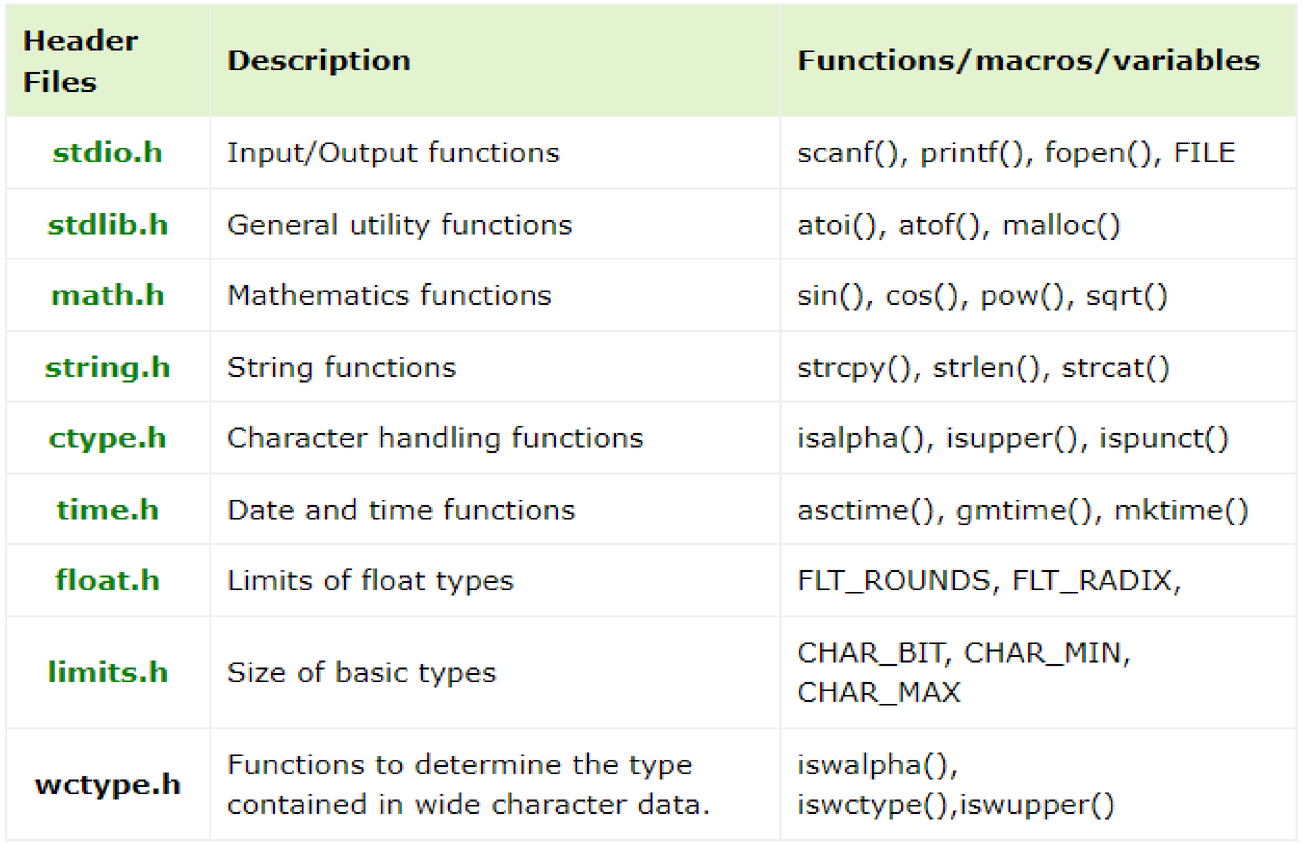
Output



```
The area of the circle is: 78.54

Process returned 0 (0x0)   executic
Press any key to continue.
```

Header Files in C Programming

- A header file has ".h" extension from which the forward declarations of one or more predefined functions, constants, macros etc can be declared.



Header Files	Description	Functions/macros/variables
stdio.h	Input/Output functions	scanf(), printf(), fopen(), FILE
stdlib.h	General utility functions	atoi(), atof(), malloc()
math.h	Mathematics functions	sin(), cos(), pow(), sqrt()
string.h	String functions	strcpy(), strlen(), strcat()
ctype.h	Character handling functions	isalpha(), isupper(), ispunct()
time.h	Date and time functions	asctime(), gmtime(), mktime()
float.h	Limits of float types	FLT_ROUNDS, FLT_RADIX,
limits.h	Size of basic types	CHAR_BIT, CHAR_MIN, CHAR_MAX
wctype.h	Functions to determine the type contained in wide character data.	iswalpha(), iswctype(),iswupper()

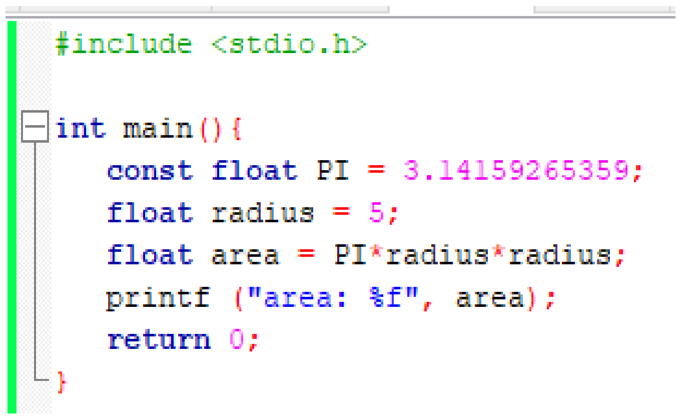
Constants (1/3)

- A constant in C is a user-assigned name to a location in the memory, whose value cannot be modified once declared.
  - This is in contrast to a variable in C, which is also a named memory location, however whose value may be changed during the course of the code.
  - For example, the value of mathematical constant PI is a high-precision floating point number 3.14159265359, and if it is likely to appear frequently, it is declared as a constant and used by its name.
- Constant can be declared in C program with either of the following two ways
  - Using the const Keyword
  - Using the #define Directive

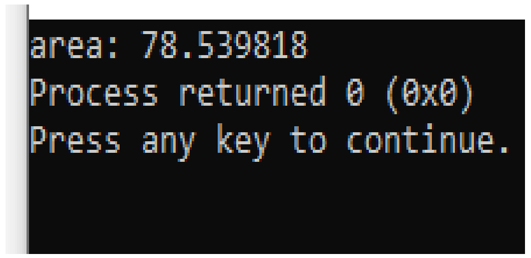
Constants (2/3)

Code

Output



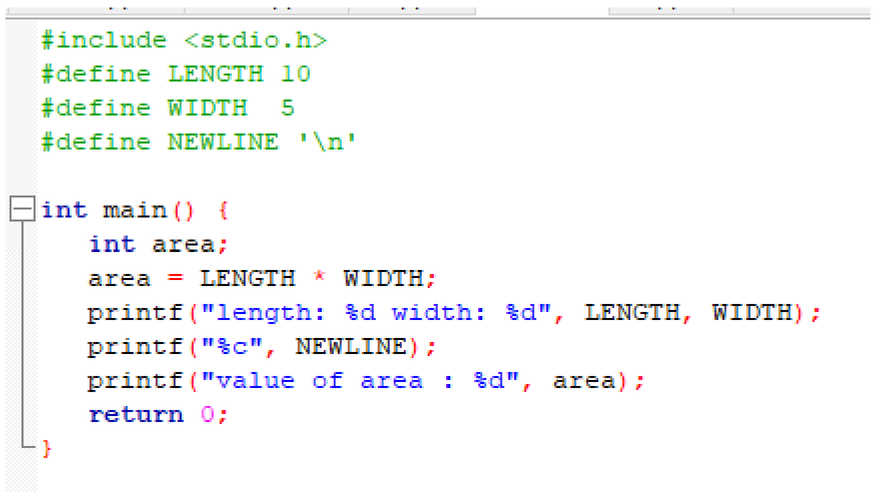
```
#include <stdio.h>

int main(){
    const float PI = 3.14159265359;
    float radius = 5;
    float area = PI*radius*radius;
    printf ("area: %f", area);
    return 0;
}
```



```
area: 78.539818
Process returned 0 (0x0)
Press any key to continue.
```

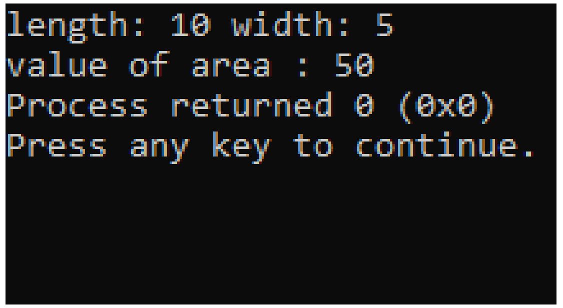
Constants (3/3)

Code



```c
#include <stdio.h>
#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main() {
    int area;
    area = LENGTH * WIDTH;
    printf("length: %d width: %d", LENGTH, WIDTH);
    printf("%c", NEWLINE);
    printf("value of area : %d", area);
    return 0;
}
```

Output



```
length: 10 width: 5
value of area : 50
Process returned 0 (0x0)
Press any key to continue.
```

Special Symbols (1/2)

- In the C programming language, special symbols (or special characters) are non-alphabetic and non-numeric characters that have special meanings and functions within the language.
  - These symbols are used to perform a variety of tasks.
- Special Symbol Examples:
  - Separating Statements: For example, a semicolon (;) is used to terminate statements.
  - Grouping Code Blocks: Curly braces ({}) are used to group multiple statements into a block.
  - Performing Operations: Symbols like +, -, *, and / are used as operators to perform mathematical and logical operations.

Special Symbols (2/2)

- Special Symbol Examples:
  - Defining Structures and Arrays: Square brackets ([]) are used for array indexing
  - Parentheses (()) are used for function calls and precedence in expressions.
  - Providing Directives: The # symbol is used in preprocessor directives (e.g., #include, #define).
- These symbols are integral to the syntax and functionality of C programs, and their correct usage is essential for writing valid and efficient code.

Operators

- C program consists of expressions that perform arithmetic and comparison operations. The special symbols from the character set of C are mostly defined as operators. For example, the well-known symbols, +, −, * and / are the arithmetic operators in C. Similarly, < and > are used as comparison operators

Signed Number

- Signed numbers in C are integers that can represent both positive and negative values.
- The most significant bit (MSB) is used as the sign bit:
  - If the sign bit is 0, the number is positive or zero.
  - If the sign bit is 1, the number is negative
- Range: For an n-bit signed integer, the range is from -2^(n-1) to 2^(n-1) - 1.
- For example, a signed 8-bit integer (signed char) has a range of -128 to 127.
- if you have a signed int (typically 32 bits in many systems):
  - - The range of values is from *-2,147,483,648* to *2,147,483,647*.
  - - The MSB being 0 means the number is non-negative (positive or zero), and when it's 1, the number is negative.

Signed Number

- Representation Example:*
- A 32-bit signed integer with binary 00000000 00000000 00000000 00000001 represents the decimal value 1.
- A 32-bit signed integer with binary 10000000 00000000 00000000 00000001 represents the decimal value -2,147,483,647.

- Signed Integers in C
  - signed int: Standard signed integer.
  - short: Usually a 16-bit signed integer.
  - long: Usually a 32-bit signed integer (64-bit on some systems).
  - long long: Usually a 64-bit signed integer.

Unsigned Number

- Unsigned numbers in C are integers that can only represent non-negative values (zero and positive numbers).
- Representation: All bits are used to represent the magnitude of the number, so there is no sign bit.
- Range: For an n-bit unsigned integer, the range is from 0 to 2^n - 1.
- For example, an unsigned 8-bit integer (unsigned char) has a range of 0 to 255.
- The range of values is from *0* to *4,294,967,295*.
- Since there's no sign bit, all 32 bits contribute to the value of the number.

Unsigned Number

- Representation Example:
  - A 32-bit unsigned integer with binary 00000000 00000000 00000000 00000001 represents the decimal value 1.
  - A 32-bit unsigned integer with binary 11111111 11111111 11111111 11111111 represents the decimal value 4,294,967,295.

- Unsigned Integers in C
  - unsigned int: Standard unsigned integer.
  - unsigned short: Usually a 16-bit unsigned integer.
  - unsigned long: Usually a 32-bit unsigned integer (64-bit on some systems).
  - unsigned long long: Usually a 64-bit unsigned integer.

Signed and Unsigned Integers
Key Differences

- Signed integers can represent negative numbers, while unsigned integers cannot.
- Usage:
  - Signed integers are used when negative values are expected, such as in temperature, altitude, or financial calculations.
  - Unsigned integers* are used when negative values do not make sense, like in counting objects, representing sizes, or indexing arrays.

ASCII

- ASCII stands for American Standard Code for Information Interchange.
    - It is a character encoding standard used to represent text in computers and other devices that use text.

- ASCII Code assigns a unique numerical value to each character, including letters, digits, punctuation marks, and control characters. These values range from 0 to 127 in the standard ASCII set.

ASCII

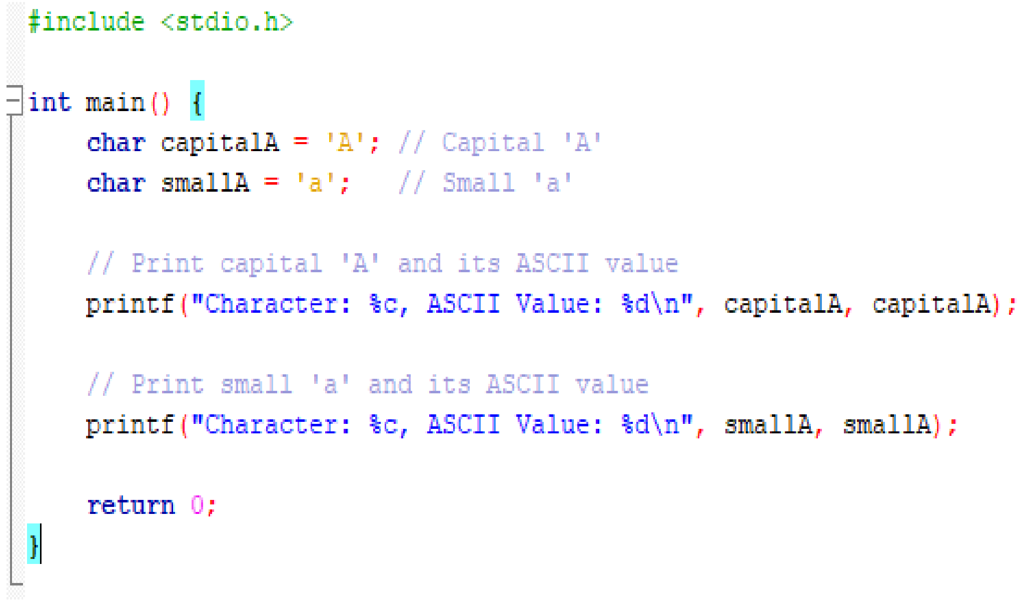
- ASCII Character Set: The ASCII character set includes-
  - Control characters (0–31): Non-printable characters used for control functions (e.g., \n for newline, \t for tab).

  - Printable characters (32–126): Characters you can see on the screen, such as letters (A-Z, a-z), digits (0-9), punctuation marks (. , ! @ etc.), and special characters.

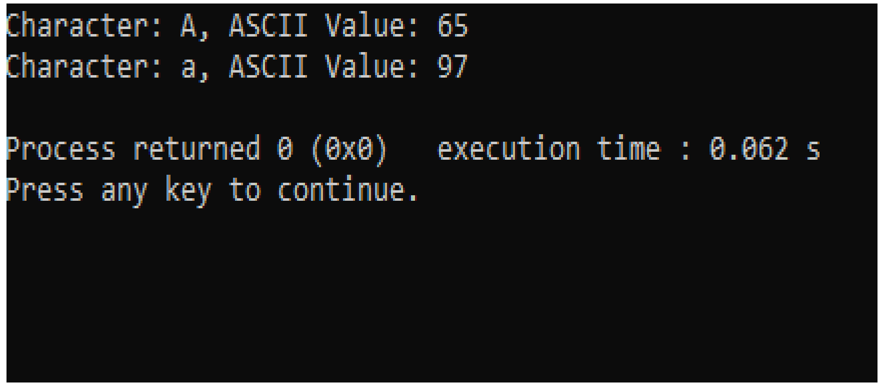
ASCII (Example)

Code

Output



```c
#include <stdio.h>

int main() {
    char capitalA = 'A'; // Capital 'A'
    char smallA = 'a';   // Small 'a'

    // Print capital 'A' and its ASCII value
    printf("Character: %c, ASCII Value: %d\n", capitalA, capitalA);

    // Print small 'a' and its ASCII value
    printf("Character: %c, ASCII Value: %d\n", smallA, smallA);

    return 0;
}
```



```
Character: A, ASCII Value: 65
Character: a, ASCII Value: 97

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

Reference

- Geeksforgeeks
- GrayCode
- Programiz.com
- TutorialsPoint

Thanks