C Keywords and Identifiers

In this tutorial, you will learn about keywords; reserved words in C programming that are part of the syntax. Also, you will learn about identifiers and proper way to name a variable.

Table of Contents

Character set

Character set is a set of alphabets, letters and some special characters that are valid in C language.

Alphabets

```
Uppercase: A B C ................................ X Y Z
Lowercase: a b c .................................. x y z
```

C accepts both lowercase and uppercase alphabets as variables and functions.

Digits

```
0 1 2 3 4 5 6 7 8 9
```

Special Characters

Special Characters in C Programming

,	<	>	.	_
(	)	;	$	:
%	[	]	#	?
'	&	{	}	"
^	!	*	/	|
-	\	~	+	

White space Characters

blank space, new line, horizontal tab, carriage return and form feed

Keywords

Keywords are predefined, reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax and they cannot be used as an identifier. For example:

```
int money;
```

Here, `int` is a keyword that indicates `'money'` is a variable of type integer.

As C is a case sensitive language, all keywords must be written in lowercase. Here is a list of all keywords allowed in ANSI C.

Keywords in C Language

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

Along with these keywords, C supports other numerous keywords depending upon the compiler.

All these keywords, their syntax and application will be discussed in their respective topics. However, if you want a brief overview on these keywords without going further, visit list of all keywords in C programming.

Identifiers

Identifier refers to name given to entities such as variables, functions, structures etc.

Identifier must be unique. They are created to give unique name to a entity to identify it during the execution of the program. For example:

```
int money;
double accountBalance;
```

Here, `money` and `accountBalance` are identifiers.

Also remember, identifier names must be different from keywords. You cannot use `int` as an identifier because `int` is a keyword.

Rules for naming identifiers

1. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
2. The first letter of an identifier should be either a letter or an underscore.

3. There is no rule on how long an identifier can be. However, you may run into problems in some compilers if identifier is longer than 31 characters.

Good Programming Practice

You can choose any name as an identifier following the rules (excluding keywords). However, give meaningful name to an identifier (variables, function names etc). It will make your and your fellow programmers life much easier.

C Programming Constants and Variables

In this tutorial, you will learn about variables, rules for naming a variable, constants and different type of constants in C programming.

Table of Contents

Variables

In programming, a variable is a container (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location. For example:

```
int playerScore = 95;
```

Here, `playerScore` is a variable of integer type. Here, the variable is assigned an integer value `95`.

The value of a variable can be changed, hence the name variable.

```
char ch = 'a';
// some code
ch = 'l';
```

Rules for naming a variable

1. A variable name can have letters (both uppercase and lowercase letters), digits and underscore only.
2. The first letter of a variable should be either a letter or an underscore.
3. There is no rule on how long a variable name (identifier) can be. However, you may run into problems in some compilers if variable name is longer than 31 characters.

Note: You should always try to give meaningful names to variables. For example: `firstName` is a better variable name than `fn`.

C is a strongly typed language. This means, variable type cannot be changed once it is declared. For example:

```
int number = 5;        // integer variable
number = 5.5;          // error
double number;         // error
```

Here, the type of `number` variable is `int`. You cannot assign floating-point (decimal) value `5.5` to this variable. Also, you cannot redefine the type of the variable to `double`. By the way, to store decimal values in C, you need to declare its type to either `double` or `float`.

Visit this page to learn more about different types of data a variable can store.

Constants/Literals

A constant is a value (or an identifier) whose value cannot be altered in a program. For example: `1,` `2.5,` `'c'` `etc.`

Here, `1,` `2.5` and `' c'` are literal constants. Why? You cannot assign different values to these terms.

You can also create non-modifiable variables in C programming. For example:

```
const double PI = 3.14;
```

Notice, we have added keyword `const`.

Here, `PI` is a symbolic constant. It's actually a variable however, it's value cannot be changed.

```
const double PI = 3.14;
PI = 2.9;            //Error
```

Below are the different types of constants you can use in C.

---

1. Integers

An integer is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:

- decimal constant(base 10)
- octal constant(base 8)
- hexadecimal constant(base 16)

For example:

```
Decimal constants: 0, -9, 22 etc
Octal constants: 021, 077, 033 etc
Hexadecimal constants: 0x7f, 0x2a, 0x521 etc
```

In C programming, octal starts with a `0`, and hexadecimal starts with a `0x`.

---

2. Floating-point constants

A floating point constant is a numeric constant that has either a fractional form or an exponent form. For example:

```
-2.0
0.0000234
-0.22E-5
```

Note: `E-5 = 10`$^{-5}$

---

3. Character constants

A character constant is created by enclosing a single character inside single quotation marks. For example: `'a'`, `'m'`, `'F'`, `'2'`, `'}'` etc;

4. Escape Sequences

Sometimes, it is necessary to use characters which cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc. In order to use these characters, escape sequence is used.

For example: `\n` is used for newline. The backslash `\` causes escape from the normal way the characters are handled by the compiler.

Escape Sequences

Escape Sequences	Character
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\\\	Backslash
\\'	Single quotation mark
\\"	Double quotation mark
\\?	Question mark
\0	Null character

5. String Literals

A string literal is a sequence of characters enclosed in double-quote marks. For example:

```
"good"                        //string constant
""                            //null string constant
"      "                      //string constant of six white space
"x"                          //string constant having single character.
"Earth is round\n"            //prints string with newline
```

6. Enumerations

Keyword `enum` is used to define enumeration types. For example:

```
enum color {yellow, green, black, white};
```

Here, `color` is a variable and `yellow`, `green`, `black` and `white` are the enumeration constants having value 0, 1, 2 and 3 respectively. For more information, visit page: C Enumeration.

You can also define symbolic constants using `#define`. To learn more, visit: C Macros

C Programming Data Types

In this tutorial, you will learn about data types and how to declare a variable in C programming.

Table of Contents

In C programming, variables (memory location) should be declared before it can be used. Similarly, a function also needs to be declared before use.

Data types simply refers to the type and size of data associated with variables and functions.

Data type can be either fundamental (provided in C compiler), or derived (derived from fundamental data types).

Fundamental data types

This section focuses on commonly used fundamental data types.

int

Integers are whole numbers that can have both zero, positive and negative values but no decimal values. Example: `0`, `-5`, `10`

We can use `int` for declaring an integer variable.

```
int id;
```

Here, `id` is a variable of type integer.

You can declare multiple variable at once in C programming. For example:

```
int id, age;
```

The size of `int` is 4 bytes (32 bits) in most compilers. Hence, it can take 2^{32} distinct states: $-2^{31}, -2^{31}+1, ..., -1, 0, 1, 2, ..., 2^{31}-2, 2^{31}-1$

, that is, from `-2147483648` to `2147483647`.

---

float and double

`float` and `double` are used to hold real numbers.

```
float salary;
double price;
```

In C, floating-point numbers can also be represented in exponential. For example:

```
float normalizationFactor = 22.442e2;
```

What's the difference between `float` and `double`?

The size of `float` (single precision float data type) is 4 bytes. And the size of `double` (double precision float data type) is 8 bytes.

---

char

Keyword `char` is used for declaring character type variables. For example:

```
char test = 'h';
```

The size of character variable is 1 byte.

---

void

`void` is an incomplete type. It means "nothing" or "no type". You can think of void as **absent**.

For example, if a function is not returning anything, its return type should be `void`.

Note that, you cannot create variables of `void` type.

---

bool

Traditionally, there was no boolean type in C. However, C99 defines a standard boolean type under `<sdbool.h>` header file. A boolean type can take one of two values, either `true` or `false`. For example:

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

int main() {
  bool a = true;

  return 0;
}
```

If you are a programming newbie, we recommend you to skip the content below for now and go on to the next chapter.

---

enum

You can create an enumerated type using `enum` keyword. An enumeration consists of integral constants. For example:

```
enum suit { club, diamonds, hearts, spades};
```

Here, a variable `suit` of `enum` type is defined. To learn more on how it works, visit: C enums

---

Complex types

In ISO C99, support for the complex type was standardized.

If you include `complex.h` header file in your program, you can use `complex` as a keyword to create and work with complex numbers. For example:

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

int main() {
     int complex z = 2 + 1 * I;
}
```

To learn more, visit: How to work with complex numbers in C?

---

short and long

If you need to use large number, you can use type specifier `long`. Here's how:

```
long a;
long long b;
long double c;
```

Here variables `a` and `b` can store integer values. And, `c` can store a floating-point number.

If you are sure, only a small integer (`[−32,767, +32,767]` range) will be used, you can use `short`.

```
short d;
```

You can always check size of a variable using `sizeof()` operator.

```
#include <stdio.h>
int main() {
   short a;
   long b;
   long long c;
   long double d;

   printf("size of short = %d bytes\n", sizeof(a));
   printf("size of long = %d bytes\n", sizeof(b));
   printf("size of long long = %d bytes\n", sizeof(c));
   printf("size of long double= %d bytes\n", sizeof(d));
```

```
    printf("Size of long double = %d bytes\n", sizeof(d));
    return 0;
}
```

signed and unsigned

In C, `signed` and `unsigned` are type modifiers. You can alter the data storage of a data type by using them. For example:

```
unsigned int x;
int y;
```

Here, the variable `x` can hold only zero and positive values because we have used `unsigned` modifier.

Considering the size of `int` is 4 bytes, variable `y` can hold values from -2^{31} to $2^{31}-1$, whereas variable `x` can hold values from `0` to $2^{32}-1$.

Derived types

Here's a list of derived types in C. These topics will be discussed in their respective chapter.

- Array type
- Pointer type
- Structure type
- Union type
- Function type
- Atomic type

C Input Output (I/O)

This tutorial focuses on two in-built functions printf() and scanf() to perform I/O task in C programming. Also, you will learn to write a valid program in C.

Table of Contents

C programming has several in-built library functions to perform input and output tasks.

Two commonly used functions for I/O (Input/Output) are `printf()` and `scanf()`.

The `scanf()` function reads formatted input from standard input (keyboard) whereas the `printf()` function sends formatted output to the standard output (screen).

Example 1: C Output

```
#include <stdio.h>        // Including header file to run printf() functi
int main()
{
     printf("C Programming");   // Displays the content inside quotation
     return 0;
}
```

Output

```
C Programming
```

How this program works?

- All valid C program must contain the `main()` function. The code execution begins from the start of `main()` function.
- The `printf()` is a library function to send formatted output to the screen. The `printf()` function is declared in `"stdio.h"` header file.
- Here, `stdio.h` is a header file (standard input output header file) and `#include` is a preprocessor directive to paste the code from the header file when necessary. When the compiler encounters `printf()` function and doesn't find `stdio.h` header file, compiler shows error.
- The `return 0;` statement is the "Exit status" of the program.

Example 2: C Integer Output

```
#include <stdio.h>
int main()
{
    int testInteger = 5;
    printf("Number = %d", testInteger);
    return 0;
}
```

Output

```
Number = 5
```

Inside the quotation of `printf()` function, there is a format string `"%d"` (for integer). If the format string matches the argument (`testInteger` in this case), it is displayed on the screen.

Example 3: C Integer Input/Output

```
#include <stdio.h>
int main()
{
    int testInteger;
    printf("Enter an integer: ");
    scanf("%d",&testInteger);
    printf("Number = %d",testInteger);
    return 0;
}
```

Output

```
Enter an integer: 4
Number = 4
```

The `scanf()` function reads formatted input from the keyboard. When user enters an integer, it is stored in variable `testInteger`.

Note the `'&'` sign before `testInteger`; `&testInteger` gets the address of `testInteger` and the value is stored in that address.

Example 4: C Floats Input/Output

```
#include <stdio.h>
int main()
{
     float f;
     printf("Enter a number: ");
// %f format string is used in case of floats
     scanf("%f",&f);
     printf("Value = %f", f);
     return 0;
}
```

Output

```
Enter a number: 23.45
Value = 23.450000
```

The format string "%f" is used to read and display formatted in case of floats.

Example 5: C Character I/O

```
#include <stdio.h>
int main()
{
     char chr;
     printf("Enter a character: ");
     scanf("%c",&chr);
     printf("You entered %c.",chr);
     return 0;
}
```

Output

```
Enter a character: g
You entered g.
```

Format string `%c` is used in case of character types.

Little bit on ASCII code

When a character is entered in the above program, the character itself is not stored. Instead, a numeric value(ASCII value) is stored.

And when we displayed that value using `"%c"` text format, the entered character is displayed.

Example 6: C ASCII Code

```
#include <stdio.h>
int main()
{
     char chr;
     printf("Enter a character: ");
     scanf("%c",&chr);

     // When %c text format is used, character is displayed in case of
     printf("You entered %c.\n",chr);

     // When %d text format is used, integer is displayed in case of ch
     printf("ASCII value of %c is %d.", chr, chr);
     return 0;
}
```

Output

```
Enter a character: g
You entered g.
ASCII value of g is 103.
The ASCII value of character 'g' is 103. When, 'g' is entered, 103 is s
```

You can display a character if you know ASCII code of that character. This is shown by following example.

Example 7: C ASCII Code

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

```
#include <stdio.h>
int main()
{
     int chr = 69;
     printf("Character having ASCII value 69 is %c.",chr);
     return 0;
}
```

Output

```
Character having ASCII value 69 is E.
```

More on Input/Output of floats and Integers

Integer and floats can be displayed in different formats in C programming.

Example #7: I/O of Floats and Integers

```
#include <stdio.h>
int main()
{

     int integer = 9876;
     float decimal = 987.6543;

     //  Prints the number right justified within 6 columns
     printf("4 digit integer right justified to 6 column: %6d\n", intege

     // Tries to print number right justified to 3 digits but the numbe
     printf("4 digit integer right justified to 3 column: %3d\n", intege

     // Rounds to two digit places
     printf("Floating point number rounded to 2 digits: %.2f\n",decimal);

     // Rounds to 0 digit places
     printf("Floating point number rounded to 0 digits: %.f\n",987.6543);

     // Prints the number in exponential notation(scientific notation)
     printf("Floating point number in exponential form: %e\n",987.6543);
     return 0;
}
```

Output

```
4 digit integer right justified to 6 column:   9876
4 digit integer right justified to 3 column: 9876
Floating point number rounded to 2 digits: 987.65
Floating point number rounded to 0 digits: 988
Floating point number in exponential form: 9.876543e+02
```

C Programming Operators

C programming has various operators to perform tasks including arithmetic, conditional and bitwise operations. You will learn about various C operators and how to use them in this tutorial.

Table of Contents

An operator is a symbol which operates on a value or a variable. For example: `+` is an operator to perform addition.

C has wide range of operators to perform various operations.

C Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus

Operator	Meaning of Operator
*	multiplication
/	division
%	remainder after division(modulo division)

Example 1: Arithmetic Operators

```
// C Program to demonstrate the working of arithmetic operators
#include <stdio.h>

int main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);

    c = a-b;
    printf("a-b = %d \n",c);

    c = a*b;
    printf("a*b = %d \n",c);

    c=a/b;
    printf("a/b = %d \n",c);

    c=a%b;
    printf("Remainder when a divided by b = %d \n",c);

    return 0;
}
```

Output

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

The operators +, - and * computes addition, subtraction and multiplication respectively as you might have expected.

In normal calculation, `9/4 = 2.25`. However, the output is 2 in the program.

It is because both variables a and b are integers. Hence, the output is also an integer. The compiler neglects the term after decimal point and shows answer 2 instead of 2.25.

The modulo operator % computes the remainder. When `a = 9` is divided by `b = 4`, the remainder is 1. The % operator can only be used with integers.

```
Suppose a = 5.0, b = 2.0, c = 5 and d = 2. Then in C programming,

a/b = 2.5  // Because both operands are floating-point variables
a/d = 2.5  // Because one operand is floating-point variable
c/b = 2.5  // Because one operand is floating-point variable
c/d = 2      // Because both operands are integers
```

Increment and decrement operators

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

Example 2: Increment and Decrement Operators

```c
// C Program to demonstrate the working of increment and decrement opera
#include <stdio.h>
int main()
{
     int a = 10, b = 100;
     float c = 10.5, d = 100.5;

     printf("++a = %d \n", ++a);

     printf("--b = %d \n", --b);

     printf("++c = %f \n", ++c);

     printf("--d = %f \n", --d);

     return 0;
}
```

Output

```
++a = 11
--b = 99
```

```
++c = 11.500000
++d = 99.500000
```

Here, the operators ++ and -- are used as prefix. These two operators can also be used as postfix like `a++` and `a--`. Visit this page to learn more on how increment and decrement operators work when used as postfix.

C Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Example 3: Assignment Operators

```
// C Program to demonstrate the working of assignment operators
#include <stdio.h>
int main()
{
    int a = 5, c;

    c = a;
    printf("c = %d \n", c);

    c += a; // c = c+a
    printf("c = %d \n", c);

    c -= a; // c = c-a
    printf("c = %d \n", c);

    c *= a; // c = c*a
    printf("c = %d \n", c);

    c /= a; // c = c/a
    printf("c = %d \n", c);
    c %= a; // c = c%a
    printf("c = %d \n", c);

    return 0;
}
```

Output

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

C Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

Example 4: Relational Operators

```
// C Program to demonstrate the working of arithmetic operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d = %d \n", a, b, a == b); // true
    printf("%d == %d = %d \n", a, c, a == c); //  false

    printf("%d > %d = %d \n", a, b, a > b); //false
    printf("%d > %d = %d \n", a, c, a > c); //false


    printf("%d < %d = %d \n", a, b, a < b); //false
    printf("%d < %d = %d \n", a, c, a < c); //true


    printf("%d != %d = %d \n", a, b, a != b); //false
    printf("%d != %d = %d \n", a, c, a != c); //true


    printf("%d >= %d = %d \n", a, b, a >= b); //true
    printf("%d >= %d = %d \n", a, c, a >= c); //false


    printf("%d <= %d = %d \n", a, b, a <= b); //true
    printf("%d <= %d = %d \n", a, c, a <= c); //true

    return 0;
}
```

Output

```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
5 > 10 = 0
5 < 5 = 0
5 < 10 = 1
5 != 5 = 0
5 != 10 = 1
5 >= 5 = 1
5 >= 10 = 0
5 <= 5 = 1
5 <= 10 = 1
```

C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning of Operator	Example

Operator	Meaning of Operator	Example
&&	Logial AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c == 5) && (d > 5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c == 5) || (d > 5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `! (c == 5)` equals to 0.

Example #5: Logical Operators

```
// C Program to demonstrate the working of logical operators

#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) equals to %d \n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) equals to %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) equals to %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) equals to %d \n", result);

    result = !(a != b);
    printf("!(a == b) equals to %d \n", result);

    result = !(a == b);
    printf("!(a == b) equals to %d \n", result);

    return 0;
}
```

Output

```
(a == b) && (c > b) equals to 1
(a == b) && (c < b) equals to 0
(a == b) || (c < b) equals to 1
(a != b) || (c < b) equals to 0
!(a != b) equals to 1
!(a == b) equals to 0
```

Explanation of logical operator program

- `(a == b) && (c > 5)` evaluates to 1 because both operands `(a == b)` and `(c > b)` is 1 (true).
- `(a == b) && (c < b)` evaluates to 0 because operand `(c < b)` is 0 (false).
- `(a == b) || (c < b)` evaluates to 1 because `(a = b)` is 1 (true).
- `(a != b) || (c < b)` evaluates to 0 because both operand `(a != b)` and `(c < b)` are 0 (false).
- `!(a != b)` evaluates to 1 because operand `(a != b)` is 0 (false). Hence, !(a != b) is 1 (true).
- `!(a == b)` evaluates to 0 because `(a == b)` is 1 (true). Hence, `!(a == b)` is 0 (false).

Bitwise Operators

During computation, mathematical operations like: addition, subtraction, addition and division are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Visit bitwise operator in C to learn more.

Other Operators

Comma Operator

Comma operators are used to link related expressions together. For example:

```
int a, c = 5, d;
```

The sizeof operator

The `sizeof` is an unary operator which returns the size of data (constant, variables, array, structure etc).

Example 6: sizeof Operator

```
#include <stdio.h>
int main()
{
     int a, e[10];
     float b;
     double c;
     char d;
     printf("Size of int=%lu bytes\n",sizeof(a));
     printf("Size of float=%lu bytes\n",sizeof(b));
     printf("Size of double=%lu bytes\n",sizeof(c));
     printf("Size of char=%lu byte\n",sizeof(d));
     printf("Size of integer type array having 10 elements = %lu bytes\n
     return 0;
}
```

Output

```
Size of int = 4 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of char = 1 byte
Size of integer type array having 10 elements = 40 bytes
```

C Ternary Operator (?:)

Ternary operator is a conditional operator that works on 3 operands.

Conditional Operator Syntax

```
conditionalExpression ? expression1 : expression2
```

The conditional operator works as follows:

- The first expression conditionalExpression is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false.
- If `conditionalExpression` is true, `expression1` is evaluated.
- If `conditionalExpression` is false, `expression2` is evaluated.

Example 7: C conditional Operator

```
#include <stdio.h>
int main(){
    char February;
    int days;
    printf("If this year is leap year, enter 1. If not enter any integer
    scanf("%c",&February);

    // If test condition (February == 'l') is true, days equal to 29.
    // If test condition (February =='l') is false, days equal to 28.
    days = (February == '1') ? 29 : 28;

    printf("Number of days in February = %d",days);
    return 0;
}
```

Output

```
If this year is leap year, enter 1. If not enter any integer: 1
Number of days in February = 29
```

Other operators such as `&` (reference operator), `*` (dereference operator) and `->` (member selection) operator will be discussed in C pointers.